B4 - Network Programming

B-NWP-400

myCHAP

A simple Challenge Handshake Authentication Protocol

1.1

{EPITECH.}

myCHAP

binary name: client
repository name: NWP_mychap_$ACADEMICYEAR
repository rights: ramassage-tek
language: C
compilation: via Makefile, including re, clean and fclean rules

- Your repository must contain the totality of your source files, but no useless files (binary, temp files, obj files,...).
- All the bonus files (including a potential specific Makefile) should be in a directory named *bonus*.
- Error messages have to be written on the error output, and the program should then exit with the 84 error code (0 if there is no error).

+ Introduction

The goal of this project is to perform a Challenge-Handshake Authentication Protocol (CHAP) to an authenticating entity.
More particularly, you have to code a client in C language and manage to interact with the authenticating entity (the server).

The complete authentication scheme (CHAP) is defined in RFC 1994. Although you don't need to perform exactly this mechanism but something similar

+ PROJECT IN-DEPTH DESCRIPTION

PARAMETERS

The program takes 3 parameters:

- `-t` (or `--target`) followed by the target of the remote authenticating entity.
- `-p` (or `--port`) followed by the port.
- `-P` (or `--password`) followed by the password that allows you to authenticate to the entity.

target can be an IPv4 address or a hostname.

PROTOCOL

1) The first message (PHASE 1) that the server is waiting for is "`client hello`".
2) The second message (PHASE 2) is sent by the server as a response, and contains a 10 bytes random challenge
3) The third message (PHASE 3) is the reponse to the challenge. You need to calculate this and send the correct answer to the server.
4) The fourth message (PHASE 4) is the answer from the server and indicates whether the authentication was successfull or not.

- For PHASE 1, if any other message other than "`client hello`" is sent to the server, the server will answer "`Protocol Mismatch`".
- The answer to the challenge is `SHA256(PHASE 2 + password)`. Remember the goal is to never send the password in cleartext.
- If more than 5 seconds have elapsed between the first message (PHASE 1) and the second message (PHASE 3), the server will answer "`Session Timeout`".
- If the answer to the challenge is correct, the server will answer with a secret (transmitted in cleartext).
- Your program, upon successfull execution **MUST** display on `stdout` the following message:
  `Secret: 'SECRET_SENT_FROM_SERVER'`
  Otherwise it must display:
  `KO`
- If the hostname we specified as argument does not resolve, you **MUST** display on `stdout` the following message:
  `No such hostname: 'nonexistent_hostname'`

sha256 is 256bits long. The answer to the challenge should be represented in hexadecimal, therefore 64 digits. Make sure your SHA256 function returns the same result as the `sha256sum` utility on Linux or any other tool online. You should send 64 bytes.

`SECRET_SENT_FROM_SERVER` is the secret sent by the server during PHASE 4, if and only if the answer to the challenge is correct

It is to be noted that it is not asked to build a mutual authentication but a one-way authentication mechanism ("half" we could say since the server is already provided)

Examples

Example 1:

```
Terminal
~/B-NWP-400> ./client -t localhost -p 4241 -P "epitech21"
Secret: 'Sh0k0b0n'
```

Example 2:

```
Terminal
~/B-NWP-400> ./client -t localhost -p 4242 -P "wrong_password"
KO
```

Example 3:

```
Terminal
~/B-NWP-400> ./client -t 12.34.56.78 -p 4243 -P "epitech42"
Secret: 'Sh0k0IsVeryb0n'
```

Example 4:

```
Terminal
~/B-NWP-400> ./client -t 94ecec46a076e83a3fac07252164af65 -p 4243 -P "epitech42"
No such hostname: '94ecec46a076e83a3fac07252164af65'
```

No need to hardcode things in your program. During evaluation the password and the secret will change

strace and wireshark are your friends; netcat can also help sometimes.

+ CONSTRAINTS

When coding network applications, normally the programmer initiates a socket in TCP or UDP and sends a message by writting something into the socket. The Kernel is responsible for filling all the other headers of the packet. That is, if we had a TCP socket and sent "Hello", the Kernel will add on top of that "Hello" message a TCP header, and on top of that header, an IPv4 header and on top of that the header of the Ethernet frame. Similarly, if we had a UDP socket and sent "Hello" into it, the Kernel will add on top of that an UDP header and on top of that an IPv4 header and again on top of that an Ethernet frame header.
You are required to code headers that are normally taken care by the Kernel.
Therefore the socket **must** be called with:
a) `AF_INET` and `SOCK_RAW` parameters, or
b) `AF_PACKET` and `SOCK_RAW` parameters (Although it is to be noted that this one might be complicated as you need to create and send an ARP packet before to get the MAC address of the router)
No matter what you choose, you need at least to code by yourself the IPv4 and UDP headers.

When coding with raw sockets a few undefined behaviours might occur. It is your duty to test that your program works with the server on localhost and remotely (server on a remote host, and client on your host)

ATTENTION: No matter how you choose to initialise the socket, it is **MANDATORY** to specify `IP_HDRINCL` with `setsockopt()`. This will ensure that the Kernel will NOT fill the IP header by itself. (It is mandatory to specify `IP_HDRINCL` no matter the OS, and even for example if the socket was created with `IPPROTO_RAW` as a third argument)

No external libs are allowed apart from libopenssl to use **only** the `SHA256_Init()`, `SHA256_Update()` and `SHA256_Final()` functions

+ LITERATURE

- CHAP
- Internet Protocol
- Image of an IPv4 header
- UDP
- Image of a UDP packet
- UDP hole punching
- NAT Behavioral Requirements for UDP
- STUN
- STUN

Of course, you do **not** need to read everything! Most of those links are here for general knowledge. That being said, you might find very useful some of those links

+ SERVER

The server is provided.
You can compile the server with:

- `-D_DEBUG` for general debugging information
- `-D_DEBUG_AUTH_STEPS` to debug authentication steps

Also, the password that the server is waiting for, can be modified by changing the `PASSWORD` macro in the `server.h` file.

A bug bounty program has been put in place for the server. If you find a vulnerability, create a patch file using diff and report it to the module leaders to retrieve your special gift.

To be able to run the client and the server on the same host, an extra very small step is needed. This is the only reason why the server is provided to you.

+ BONUS

- Able to run multiple clients (simultaneously) on same host.

Focus mostly on the main program, and not the bonus, even if it is very simple to code

+ FAQ

- Q: What is the objective of this project ?
  A: It is a small (but good) introduction into low level network programming.
  Mainly reimplement some headers (in Userspace) and learn how computers communicate between them.

- Q: The program that it is asked to code differs quite a lot from the CHAP rfc. Why is that so ?
  A: To make the project less complicated.

- Q: Rather than building a one-way authentication, why not build a mutual authentication since it is much stronger ?
  A: To make the project less complicated.

- Q: I managed to send the first packet and receive the challenge, but when I am sending the second message (PHASE 3) I get "Protocol Mismatch". Why ?
  A: That means that the server understand this as a separate connection. If you are getting this error but send your second message under 5 second from the first one, you should look at the packet itself. Ask yourself, what do you need to conserve in the packet for the server (or even the NATs located between the sender and the destination) to understand this as the same connection as your first packet ?

- Q: I have a server and on its side, when I am listening to the frames, I can see that the source port of my packet that was sent from the client has changed. Why ?
  A: This is because your packet passed through at least 1 NAT. Read the literature for more information.

- Q: My program works, but **sometimes** like 1 out of 30 times it fails. Why ?
  A: Well, it is UDP, what do you expect ?

- Q: Why not in TCP ?
  A: Several reasons:
  1) First of all it might be difficult as a first network program as you need to implement the 3 way handshake before having a communication in TCP.
  2) The TCP header is more complicated than UDP and has several fields to learn.
  3) And the most important, the Kernel will itself answer to TCP responses. Apart from the fact that it might be difficult to force you to disable a few things in your Kernel, doing so will also render your computer almost unusable.

- Q: I am coding on a *BSD. I can send packets from my raw socket but not receive them. Why ?
  A: Have a look here. Remember your program must work on the dump.

- Q: I stopped coding the project. Instead I am having sooo much fun by sending multiple packets in a loop to 1 remote destination at a rate of approximately 10 000 - 100 000 pps. Am I disturbing someone ?
  A: I do not know. Wait a bit, you might receive a call from your ISP or a network administrator. Code responsibly, risks are real when sending stuff over the Internet network.