TD 3: capabilities / credentials

1 PAM

This must be run on a CREMI machine.

1.1 Services

Open `/etc/pam.d/login` (used when logging in on the keyboard/screen) and `/etc/pam.d/sshd` (used when logging in through ssh), so as to be able to compare them to see the difference between the two cases.

1.1.1 account

Notice that for the `account` part, they basically just include `common-account`, we will see that common part later.

1.1.2 auth

Notice that for the `auth` part, they include `common-auth`, but for `login`, there is an additionnal `pam_securetty.so` module to be able to handle differently the authentication on a TTY that is considered "secure" (i.e. the physical keyboard/screen).

1.1.3 password

They basically just include `common-password`

1.1.4 session

There are various modules involved in both `login` and `sshd`, they are mostly the same.

For `login`, there is an additionnal `pam_group.so` module to tune the assigned groups. The comment mentions `/etc/security/group.conf`, read it. See that it brings a list of gid to be automatically added for users logging in through the session manager (kdm or gdm). They correspond to being able to use the hardware around the physical machine, which we don't want to allow for remote users.

1.2 Modules

Let's look more closely at the common files.

1.2.1 account

Open `common-account`.

As the comment says, this only uses the `pam_unix.so` module to deny access for accounts that are expired according to `/etc/shadow`. Otherwise it just returns success.

1.2.2 auth

Open `common-auth`.

See that this was tuned for the needs of CREMI: it uses the `pam_unix.so` for local accounts (e.g. root), `pam_krb5.so` for authentication via a Kerberos ticket, and `pam_ldap.so` for authentication via LDAP (i.e. your usual login/password way).

1.2.3 password

Open `common-password`.

There is no `pam_ldap.so` module here, but that could have been used, to let users update their LDAP password simply from the `passwd` command.

1.2.4 session

Not much interesting to see in `common-session`.

2 Credentials

Let's now program a bit. First let's set up a client/server relation.

2.1 Client/server sockets

- Get `https://dept-info.labri.fr/~thibault/SecuSys/td3.tgz`
- Run `make`
- In one terminal, run `strace ./server`
- In another terminal, run `strace ./client`

See that the client reads what the server wrote, good!

2.2 Getting the peer creds

Now we want the server to check what identity the client is running under. This can be achieved easily with `SO_PEERCRED`.

- Read the section of `man 7 unix` that talks about it.
- In the server, between accepting the connection and the write, call `getsockopt` on the `client` socket, with the `SOL_SOCKET` level and the `SO_PEERCRED` option. As described in the `unix` man-page, you have to pass the address of a `struct ucred` to get the content of the option (and the address of a variable that contains the size of that structure).
- You can now print the content of the `struct ucred` (the content is described in `man 7 unix`, in the section about `SCM_CREDENTIALS`, use `%d` to print the fields)
- On the same machine, run the server as one user, then `chmod 777 /tmp/mysocket` to open access to other users, and then run the client as another user, to check that the credentials are correct.

Note that this only works with the UNIX local sockets, for which the OS has a direct relation between the client and the server.

2.3 (If you have time) Sending a file descriptor

If you have time, you can try to pass a file descriptor over a socket.

This can be useful when the server is run under a given identity which allows it to open the file, and hands the file descriptor to the client under various conditions.

This can be achieved with `SCM_RIGHTS`

- Read the section of `man 7 unix` that talks about it.
- Read the second part of the examples section of `man 3 cmsg`, it shows how to stuff the file descriptor numbers into a `msghdr`.
- Make the server open a file.
- In the server, replace the `write` call with a `sendmsg` call, which allows to not only pass the string (through the `iovec` structure), but also the file descriptor (through the `cmsg`).
- In the client, replace the `read` call with a `recvmsg` call. Note that you have to initialize the `msg_iov` and `msg_iovlen` fields to set up the reception of the string, and the `msg_control` and `msg_controllen` fields to set up the reception of the file descriptor (as a new file descriptor number).
- In the client you can now go through the `cmsg` (as shown in the examples section of `man 3 cmsg`) to get the file descriptor.
- Try to read from it! (run through `strace` to check that everything is going as planned).

Note that the file descriptor number is different in the client and in the server: indeed it's not the number that is transferred, it is the opened file description within the kernel (i.e. the `struct file *` in the kernel), referenced in the client under a new file descriptor number.