Segurança de Sistemas Informáticos

Trabalho Prático nº 2

Universidade do Minho, Departamento de Informática
Diogo Abreu e Rodrigo Monteiro
{100646, 100706}@alunos.uminho.pt

Grupo 21

1. Introdução

Neste trabalho implementamos um serviço de comunicação entre utilizadores locais num ambiente Linux. O objetivo principal é criar uma plataforma que permita o envio e a leitura de mensagens entre utilizadores, além da gestão de utilizadores e de grupos privados de comunicação, mantendo segurança e confidencialidade.

2. Arquitetura funcional

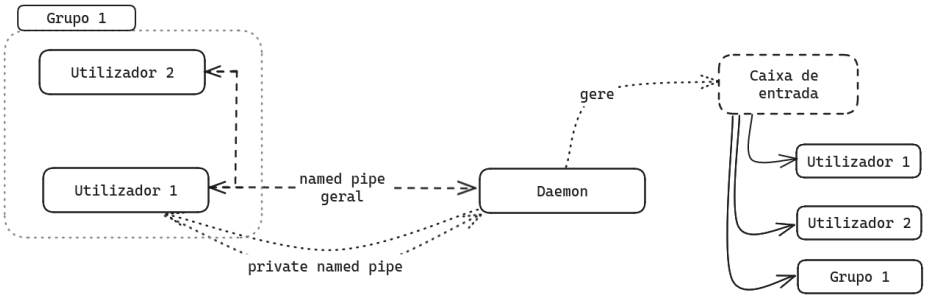

Figura 1: Esboço da arquitetura funcional

2.1. Inicialização do *daemon*

```
[Unit]
Description=Concordia
After=network.target

[Service]
Type=simple
ExecStart=/home/.../s
Restart=always

[Install]
WantedBy=multi-user.target
```

Listing 1: `concordia.service`

- Colocar o `concordia.service` no `/etc/systemd/system` (`sudo concordia.service /etc/systemd/system`);
- Nessa diretoria, executar `sudo systemctl enable concordia.service`;
- Depois, `sudo systemctl start concordia.service`.

2.2. Mecanismos de comunicação

Os utilizadores do serviço utilizam um ficheiro executável para comunicar com o *daemon* – um processo servidor que executa em pano de fundo. Esse ficheiro comunica com o servidor através de *named pipes*. Primeiramente é efetuado um "handshake", depois são criados dois *named pipes* privados para se obter uma comunicação bidirecional que são usados durante o resto do programa.

A maior parte dos dados enviados iniciam com uma estrutura `header_private`, que possui o tipo e o tamanho dos dados que se seguem, como, por exemplo, a estrutura `activate_user`.

```
typedef struct header_private {
  size_t size;
  REQUEST_TYPE type;
} HEADER_PRIVATE;

typedef struct activate_user {
  int user_id;
  int group_id;
} ACTIVATE_USER;
```

Listing 2: `Estrutura dos dados`

3. Segurança

Cada utilizador e cada grupo possui uma pasta, que serve como uma caixa de entrada, onde ficam guardados ficheiros em formato binário que representam as mensagens recebidas.

3.1. Inicialização

```
sudo rm -rf /tmp/mail_box
sudo mkdir /tmp/mail_box
sudo chown root:app /tmp/mail_box
sudo chmod 770 /tmp/mail_box
sudo chown root:root s
sudo chmod u+s s
sudo setcap cap_chown,cap_setuid,cap_setgid+ep s
```

Listing 3: `Permissoes`

- Todos os utilizadores que utilizem o serviço tem de fazer parte do grupo `app`;
- A pasta `mail_box` tem como *owner* a `root`, e como grupo `app`

- Não é possível aceder à pasta através do conjunto de permissões `other`
- O ficheiro executável do servidor/ daemon tem permissões de `root`, e, para além disso, são adicionadas `capacities` de modo a que este possa mudar `user ids`, `group ids`, ou `directory ids` em *runtime*.

3.2. Ativar e desativar utilizadores

```
if (setegid(APP_GROUP_ID) == -1) { send_status(fd, ERROR); return; }
if (seteuid(user_id) == -1) { send_status(fd, ERROR); return; }
if (mkdir(path, 0700) == -1) { send_status(fd, ERROR); return; }
// ...
```

Listing 4: `ativar()`

O processo muda, primeiro, o seu `group_id` (para `app`) e depois o seu `user_id` para o `id` do utilizador que chamou o comando. Com essa configuração, cria uma pasta com permissões apenas para o `owner`. Por fim, ids sao restaurados para os seus valores anteriores. Assim, apenas esse utilizador terá permissões em relação a essa diretoria.

3.3. Criar e remover grupos

Para criar e remover grupos, foi necessário escrever escrever no ficheiro `/etc/group` – sendo necessária a permissão de `root`, concedida na inicialização com foi visto anteriormente. No entanto, antes da escrita, é necessário verificar se o utilizador que chamou o comando para eliminar o grupo e ´o dono do grupo; caso não seja, não o pode remover e é devolvido um erro.

```
if ((int) file_stat.st_uid != owner_id) {
  send_status(fd, NO_PERMISSION);
  return;
}

FILE *file, *tempFile;
char line[MAX_LINE_LENGTH];
char searchString[10];
sprintf(searchString, ":%d:", group_id);

file = fopen("/etc/group", "r");
// ...
```

Listing 5: `Permissoes`

Para criar a pasta é utilizado o `user_id` e o grupo `app` (possui permissões para escrever). Depois de ser criada, o grupo desta pasta é substituído pelo `group_id` recebido.

3.4. **Envio e listagem de mensagens**

Cada mensagem ´e guardada na pasta do utilizador ou grupo com o seguinte *path*: “n.bin”, em que n ´e o índice da mensagem (obtido atrav´es da função `count_files_in_directory`).

Configurações para um utilizador

```
user_id = msg.dest;
group_id = APP_GROUP_ID;
```

Configurações para um grupo

```
user_id = original_euid;
group_id = msg.dest;
```

Para a listagem, é gerado um array de `SEND_MSG`, e depois é iterado, sendo enviada uma mensagem com a respetiva struct a cada iteração.

4. **Conclusões**

Por fim, apesar de faltarem algumas funcionalidades na implementação, achamos que conseguimos aplicar o conhecimento da segunda parte desta unidade curricular de forma adequada.