Semana 3

Controlo de Acesso ao Sistema de Ficheiros em Linux

Para cada uma das secções (1 a 4) deste guião é proposto um conjunto de exercícios. A resolução dos mesmos consiste numa série comandos *UNIX* que devem ser colocados num script *bash*. O nome do script deverá seguir o formato `secX.sh`, onde `X` corresponde ao número da secção em questão. Convém relembrar que para correr um script, é necessário adicionar as respetivas permissões de execução do mesmo. Como sugestão, podem utilizar o seguinte excerto para a inicialização dos scripts:

```
#!/bin/bash

# Exercício 1
comando A

# Exercício N
comando B
comando C

...
```

NOTA: Dado que neste guião se pretende trabalhar com a noção de utilizadores e grupos do sistema, sugere-se a utilização de ambientes virtuais (ex. VMs) de modo a evitar potenciais problemas indesejados nas vossas máquinas. Uma possível sugestão para poderem ter múltiplos ambientes virtuais leves e descartáveis passa por utilizarem o multipass. Podem instalar o mesmo através da página oficial ou através do vosso gestor preferido (`brew`, `snap`, etc). Após instalado, podem criar instâncias utilizando o comando `multipass launch --name NOME` e conectarem-se a estas através do comando `multipass shell NOME`.

1. Utilizador, Grupo e Permissão

Contextualização

- Noção de utilizador, grupo principal e grupos secundários
- O utilizador root
- Noção de utilizador humano e virtual
- Noção de ficheiro, diretoria e de i-node
- Estrutura e semântica das permissões definidas em ficheiros
- Estrutura e semântica das permissões definidas em diretorias
- Controlo de acesso ao longo de um caminho para um ficheiro ou diretoria

Objetivos

- Definição e experimentação das permissões definidas para o utilizador dono de um ficheiro
- Definição e experimentação das permissões definidas para o utilizador dono de uma diretoria
- Experimentação do controlo de acesso em cada uma das componentes de um caminho para um ficheiro ou diretoria

Comandos Relevantes

- chmod
  - Tenha em conta que as permissões podem ser expressas simbolicamente e em octal
- chown
  - Tenha em conta que também pode ser usado para definição do grupo proprietário de um ficheiro ou diretoria
- chgrp
  - Tenha em conta que o comando está restrito aos grupos de que utilizador comum já faz parte
- umask
  - Tenha em conta que o valor definido retira permissões definidas por omissão (0666 ou 0777)
  - Tenha em conta que o valor pode ser redefinido

Exercícios

1. Crie os ficheiros `lisboa.txt`, `porto.txt` e `braga.txt` e inclua um excerto de texto em cada um destes ficheiros.
2. Execute o comando necessário para visualizar as permissões referentes ao ficheiro `lisboa.txt`.
3. Altere as permissões do ficheiro `lisboa.txt` de modo que o dono (*owner*), o grupo (*group*), e restantes utilizadores (*other*) possuam permissões de leitura e escrita.
   - Podem ser utilizadas permissões no formato numérico ou utilizando caracteres.
4. Altere as permissões do ficheiro `porto.txt` de modo que o dono possua permissões de leitura e execução, mas não possua permissões de escrita.
5. Altere as permissões do ficheiro `braga.txt` de modo que apenas o dono possua permissões de leitura.
6. Crie as diretorias `dir1` e `dir2` e execute os comandos necessários para visualizar as permissões referentes às mesmas.
7. Remova todas as permissões de execução da diretoria `dir2`, exceto para o dono.

2. Gestão de Utilizadores e de Grupos

Contextualização

- Estrutura e função dos ficheiros que sustentam a base de dados de utilizador e de grupos de utilizadores (/etc/passwd, /etc/groups)
- A função dos ficheiros sombra (/etc/shadow, /etc/gshadow)
- Utilização do comando `sudo`

Objetivos

- Exercitar a gestão de utilizadores e de grupos de utilizadores
- Refletir sobre as permissões associadas a estes ficheiros

Comandos Relevantes

- id, groups
- sudo (para uso com os comandos abaixo, revistar também os comandos chown e chgrp)
- adduser, deluser, usermod
- groupadd, groupdel, groupmod, groupmems
- passwd, gpasswd
- su (para inciar uma sessão associada a um outro utilizador)

Exercícios Propostos

0. Observe o conteúdo dos ficheiros `/etc/passwd` e `/etc/group`.
1. Crie um utilizador para cada membro da equipa.
2. Crie o grupo `grupo-ssi` contendo todos os elementos da equipa, e crie um segundo grupo `par-ssi` contendo apenas 2 elementos da equipa.
3. Observe novamente o conteúdo dos ficheiros `/etc/passwd` e `/etc/groups`. Observou alguma diferença?
    - Neste exercício, a resposta pode ser dada sob a forma de um comentário no script.
4. Altere o dono do ficheiro `braga.txt` para um dos utilizadores criados no ponto `1.`.
5. Leia o conteúdo do ficheiro `braga.txt`.
6. Inicie sessão com o utilizador especificado em `3.`.
7. Execute os comandos `id` e `groups` e comente o resultado impresso no terminal.
    - Neste exercício, a resposta pode ser dada sob a forma de um comentário no script.
8. Leia o conteúdo do ficheiro `braga.txt`. Observou alguma diferença?.
    - Neste exercício, inclua um comentário com uma breve análise do comportamento obtido.
9. Mude para diretoria `dir2` e comente o resultado.
    - Neste exercício, a resposta pode ser dada sob a forma de um comentário no script.

3. Utilizador Real vs. Efetivo e Elevação de Privilégio

Contextualização

- Noção de utilizador (e grupo) real e efetivo associados à execução de um processo
- setuid e setgid como permissões que permitem a redefinição do utilizador efetivo

Objetivos

- Definir e experimentar as consequências do uso das permissões setuid e setgid

Comandos Relevantes

- su, sudo

Exercícios Propostos

- **Nota**: Execute o comando `exit` para terminar a sessão do utilizador previamente escolhido.

1. Crie um programa binário executável que imprima o conteúdo de um ficheiro de texto cujo nome é passado como único argumento da sua linha de comando (ou erro caso não o consiga fazer).
2. Crie o utilizador `userssi`.
3. Altere o dono do executável criado e do ficheiro `braga.txt` para `userssi`.
4. Execute o programa criado passando como argumento `braga.txt`.

5. Defina a permissão de `setuid` para o ficheiro executável.
6. Repita o ponto `4.` e comente o resultado.
    - Neste exercício, a resposta pode ser dada sob a forma de um comentário no script.

4. Listas Estendidas de Controlo de Acesso

Contextualização

- Estrutura das listas estendidas de controlo de acesso em Linux

Objetivos

- Compreender o uso de ACLs estendidas como forma de superação de algumas das limitações do controlo de acesso tradicional ao sistema de ficheiros.

Comandos

- setfacl, getfacl

Exercícios Propostos

- **Nota**: De modo a poder utilizar as funcionalidades de ACLs, será necessário instalar as dependências utilizando o comando `sudo apt install acl`.

1. Execute o comando `getfacl` para o ficheiro `porto.txt`.
2. Utilizando os mecanismos de ACL estendida, defina as permissões de escrita para o grupo `grupo-ssi` relativamente ao ficheiro `porto.txt`.
3. Execute o comando `getfacl` para o ficheiro `porto.txt` e comente eventuais diferenças face ao ponto `1.`.
    - Neste exercício, a resposta pode ser dada sob a forma de um comentário no script.
4. Inicie sessão como um dos utilizadores do grupo criado, e altere o conteúdo do ficheiro `porto.txt`. De seguida, tente ler o conteúdo que acabou de escrever no ficheiro. Comente o resultado.
    - Neste exercício, a análise do resultado pode ser dada sob a forma de um comentário no script.

5. Capabilities do Linux

Contextualização

- Limitações do modelo tradicional de privilégios em Unix/Linux (utilizador comum vs. root)
- Noção de capabilities como forma de decomposição granular dos privilégios do root
- Diferença entre capabilities de processo e de ficheiro
- Estrutura das capabilities: Permitted, Inheritable, Effective, Bounding e Ambient sets
- Riscos de segurança associados ao uso inadequado de setuid root vs. capabilities

Objetivos

- Compreender a necessidade de um modelo de privilégios mais granular
- Identificar e listar `capabilities` associadas a processos e ficheiros
- Atribuir `capabilities` específicas a executáveis evitando o uso de `setuid root`
- Experimentar o princípio do menor privilégio através de capabilities

Comandos Relevantes

- getcap - listar capabilities de ficheiros
- setcap - definir capabilities em ficheiros
- capsh - testar e explorar capabilities
- getpcaps - listar capabilities de processos em execução

Capabilities Comuns

Algumas das capabilities mais relevantes incluem:

- **CAP_NET_BIND_SERVICE** - permite binding a portas TCP/UDP < 1024
- **CAP_NET_RAW** - permite uso de sockets RAW e PACKET
- **CAP_DAC_OVERRIDE** - permite bypass das permissões de leitura/escrita/execução
- **CAP_DAC_READ_SEARCH** - permite contornar as permissões de leitura e acesso a diretorias
- **CAP_CHOWN** - permite alteração arbitrária de donos de ficheiros
- **CAP_SETUID** / **CAP_SETGID** - permite manipulação de UIDs/GIDs de processos
- **CAP_SYS_ADMIN** - várias operações administrativas (bastante abrangente)
- **CAP_KILL** - permite envio de sinais para processos de outros utilizadores

Lista completa: `man capabilities`

Exercícios Propostos

1. Liste todas as `capabilities` disponíveis no sistema usando o comando `capsh --print`.
2. Crie um programa em C (`webserver.c`) que tente fazer binding à porta 4050. O programa deve:
    - Aceitar como argumento o número da porta
    - Tentar criar um socket TCP e fazer bind à porta especificada
    - Imprimir mensagem de sucesso ou erro

    Exemplo de estrutura básica:

    ```
     #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>
    #include <sys/socket.h>
    #include <netinet/in.h>
    #include <unistd.h>

    int main(int argc, char *argv[]) {
        if (argc != 2) {
            fprintf(stderr, "Usage: %s <port>\n", argv[0]);
            return 1;
        }

        int port = atoi(argv[1]);
        int sockfd = socket(AF_INET, SOCK_STREAM, 0);

        if (sockfd < 0) {
            perror("Error when creating socket");
            return 1;
        }

        struct sockaddr_in addr;
        memset(&addr, 0, sizeof(addr));
        addr.sin_family = AF_INET;
        addr.sin_addr.s_addr = INADDR_ANY;
        addr.sin_port = htons(port);

        if (bind(sockfd, (struct sockaddr*)&addr, sizeof(addr)) < 0) {
            perror("Errr on bind");
            close(sockfd);
            return 1;
        }

        printf("Success: binded to port %d\n", port);
        close(sockfd);
        return 0;
    }
    ```

3. Execute o programa anterior, mas desta vez sobre a porta 80.
    - Qual o resultado obtido, e qual o motivo?
    - Como poderiam ser utilizadas as `capabilities` para executar o programa?

Referências Relevantes

- multipass docs
- permissões UNIX
- Gestão de utilizadores e grupos
- setuid / setgid
- ACLs