Laboratório 1.6: Segurança no Linux, Usuários, Senhas, Permissões, SELinux

Introdução à Segurança do Linux

O Linux utiliza duas formas principais de segurança:

- *Discretionary Access Control* (DAC)
- *Mandatory Access Control* (MAC)

O *Discretionary Access Control* (DAC) é a forma mais comum e conhecida de segurança no Linux e em outros sistemas. Ele é baseado no uso de login, senhas e permissões de acesso a arquivos, diretórios e processos.

Já o *Mandatory Access Control* (MAC) é um pouco menos conhecido e se baseia em limitar o acesso de programas/processos a um conjunto mínimo de recursos (arquivos, processos, variáveis de ambiente), bloqueando todo acesso a recursos não explicitamente permitido. As duas principais implementações de MAC no Linux é o SELinux (usado pelo Android) e o AppArmor (usado no Ubuntu e no Mint).

Neste laboratório, exploraremos mais o DAC. Fique tranquilo pois, quando formos explorar o AOSP/Android, praticaremos bastante o MAC através do SELinux.

Atenção: neste laboratório, alguns comandos serão executados como `root` (administrador), usando o `sudo`. Apesar de ser muito difícil para um usuário normal conseguir tirar um sistema Linux do ar, é **extremamente fácil** para o usuário `root` executar um comando errado e danificar o sistema. O Linux confia cegamente e executa todos os comandos do `root`, sem verificar se eles estão corretos ou analisar se eles podem prejudicar o sistema.

Portanto, pedimos um cuidado especial ao executar os comandos começando por `sudo`. Na dúvida, entre em contato com o professor ou monitor.

Outra observação importante é que alguns conhecimentos passados neste laboratório podem ser usados em práticas consideradas antiéticas. Todas as informações são passadas no intuito único de praticar e entender os conceitos passados e não devem ser usadas para práticas indevidas. Pede-se um comportamento mais responsável no decorrer deste laboratório em especial.

1. Usuários no Linux

Os usuários no Linux, são definidos no arquivo `/etc/passwd`:

```
$ tail -5 /etc/passwd                          # Mostra as últimas 5 linhas do arquivo
```

```
tcpdump:x:108:113::/nonexistent:/usr/sbin/nologin
sshd:x:109:65534::/run/sshd:/usr/sbin/nologin
landscape:x:110:115::/var/lib/landscape:/usr/sbin/nologin
pollinate:x:111:1::/var/cache/pollinate:/bin/false
marcos:x:1000:1000:,,,:/home/marcos:/bin/bash
```

Além dos usuários normais (você), tem-se diversos outros usuários de sistema. Um deles é o *root* que é o administrador principal de qualquer sistema Linux/Unix:

```
$ head -1 /etc/passwd                          # Mostra a primeira linha do arquivo (usuário root)
```

```
root:x:0:0:root:/root:/bin/bash
```

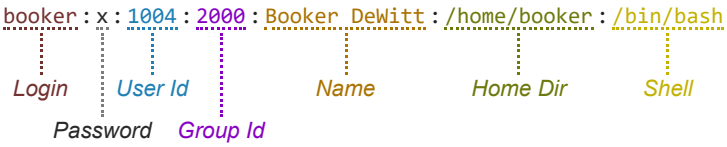
Outros usuários de sistema incluem usuários criados especificamente para rodar um determinado serviço/*daemon*:

```
$ cat /etc/passwd | grep "mail\|www-data\|syslog\|sshd"
```

```
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
syslog:x:104:110::/home/syslog:/usr/sbin/nologin
sshd:x:109:65534::/run/sshd:/usr/sbin/nologin
```

O motivo de se executar alguns serviços usando usuários próprios é que, caso eles sejam invadidos, o atacante não terá acesso ao sistema como *root*, e sim como o usuário normal do próprio *daemon*, impedindo ou dificultando o acesso a informações privilegiadas. Algo muito parecido é usado pelo Android para proteger os dados de um app e impedindo que eles sejam acessados por outros apps, como veremos futuramente.

Cada linha do arquivo `/etc/passwd` possui 7 campos separados por `:` (dois pontos). A figura a seguir mostra um exemplo de uma linha:



A tabela a seguir descreve cada um dos campos:

Campo #	Nome	Exemplo	Descrição
1	*Login*	`booker`	*Login*, ou *username*, do usuário.
2	*Password*	`x`	Antigamente, ele continha a senha criptografada do usuário. Hoje as senhas são colocadas no arquivo `/etc/shadow` e este campo tem apenas um `x` para manter compatibilidade.
3	User Id	`1004`	Número de identificação do usuário. Cada usuário precisa ter um número único, pois é esse número que é colocado nas permissões dos arquivos e diretórios.
4	*Group Id*	`2000`	Id do grupo principal do usuário. Veremos o arquivo de grupos `/etc/group` mais adiante.
5	*Name*	`Booker DeWitt`	Nome completo do usuário, podendo conter também outros comentários como local de trabalho, telefone, etc.
6	*Home Dir*	`/home/booker`	Diretório *Home* do usuário.
7	*Shell*	`/bin/bash`	Qual interpretador de comandos executar quando o usuário fizer login em modo texto.

A vantagem de se ter arquivos deste tipo, ou seja, dados em uma única linha com campos separados por um caractere, é que podemos usar os diversos comandos do Linux para obter informações. Por exemplo, para saber a quantidade de usuários cadastrados:

```
$ cat /etc/passwd | wc -l
```

```
31
```

Para listar apenas o login dos usuários, ordenados, em uma única linha:

```
$ cat /etc/passwd | cut -f1 -d: | sort | tr "\n" " " ; echo
```

```
_apt backup bin daemon games gnats irc landscape list lp mail man marcos messagebus news nobody
pollinate proxy root sshd sync sys syslog systemd-network systemd-resolve systemd-timesync tcpdump tss
uucp uuidd www-data
```

Talvez, o comando anterior precise de uma explicação mais detalhada. O `cat` manda o conteúdo do arquivo para a saída padrão, que é enviado para o `cut`, que remove tudo após o primeiro `:` e manda o resultado para o comando `sort`, que ordena as linhas da entrada padrão e joga para a saída que, por fim, é enviado para o comando `tr` que substitui todos os caracteres `\n` (nova linha) pelo caractere de espaço. No final, usamos o caractere `;` para separar o comando anterior do próximo, que é o `echo` sem parâmetros, que imprime uma linha nova.

Criando um Novo Usuário

Apesar de ser perfeitamente possível criar um usuário manualmente, editando os arquivos e criando o diretório *home*, não há necessidade de fazer isso. O Linux possui os comandos `adduser` e `useradd` para facilitar a criação de usuários. O mais completo e amigável dos dois é o `adduser`.

Vamos criar um usuário. Este usuário será usado para brincar e explorar a senha criptografada na próxima seção. Como, futuramente, você irá digitar a senha dele no terminal e em arquivos, por favor, **não use uma senha pessoal**! Use senhas fáceis como `Teste123`.

Para adicionar um usuário, é necessário ser *root*, por isso usaremos o `sudo` no comando abaixo. Note que o `sudo` pode pedir a senha do seu usuário atual antes de realmente executar o comando especificado. Se você digitou a sua senha recentemente, ele não pede novamente.

```
$ sudo adduser <LoginEscolhido>              # Substitua <LoginEscolhido> pelo LOGIN do novo usuário (sem <>)
```

```
Adding user `masp' ...
Adding new group `masp' (1001) ...
Adding new user `masp' (1001) with group `masp' ...
Creating home directory `/home/masp' ...
Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for masp
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] y
```

Veja se o seu usuário foi criado no `/etc/passwd`:

```
$ cat /etc/passwd | grep '^masp:'
```

```
masp:x:1001:1001:,,,:/home/masp:/bin/bash
```

2. Senhas no Linux

Para mudar a sua senha no Linux, execute o comando `passwd`. Ele pedirá a senha atual e, em seguida, a nova.

As senhas dos usuários ficam armazenadas no arquivo `/etc/shadow`:

```
$ cat /etc/shadow
```

```
cat: /etc/shadow: Permission denied
```

Se você executou o comando anterior como usuário normal, deve ter notado que o acesso foi negado, por motivos óbvios. Esse foi o motivo da senha ter saído do `/etc/passwd`, permitindo o acesso aos dados do usuário, mas não à senha. Para ver as senhas, execute o mesmo comando como *root*:

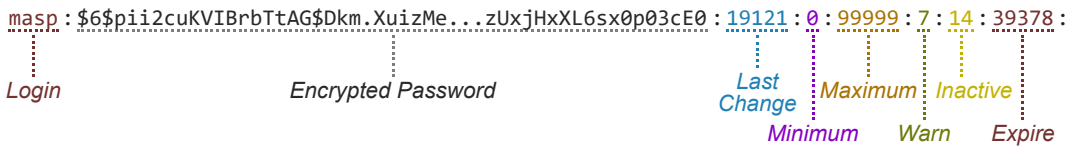
```
$ sudo cat /etc/shadow
```

Note como muitos usuários não possuem senha (o segundo campo é `*` ou `!`), ou seja, não é possível fazer login através deles. Para ver a senha do seu usuário de teste, execute:

```
$ sudo cat /etc/shadow | grep '^masp:'
```

```
masp:$6$pii2cuKVIBrbTtAG$Dkm.XuizMeqvTuL5/bwHuUQs/g0QXu6aKehj/kdbLr86Djr2WdfGASLyokgz7jg62XcAzUxjHx
```

Cada linha do arquivo `/etc/shadow` possui 8 campos separados por `:` (dois pontos). A figura a seguir mostra um exemplo, destacando os campos:



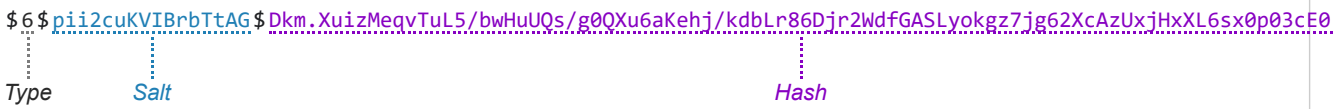
A tabela a seguir descreve cada um dos campos:

Campo #	Nome	Exemplo	Descrição

1	*Login*	`masp`	*Login* do usuário conforme existente no `/etc/passwd`.
2	*Password*	`$6$pii2cuKVIBrbT...3cE0`	A senha criptografada do usuário.
3	*Last Change*	`19121`	Data da última modificação da senha, definida em quantidade de dias após 01/01/1970.
4	*Minimum*	`0`	Quantidade mínima de dias permitida entre troca de senhas.
5	*Maximum*	`99999`	Quantidade máxima de dias que o usuário pode ficar sem trocar a senha.
6	*Warn*	`7`	Número de dias que o usuário deve ser avisado antes da senha expirar.
7	*Inactive*	`14`	Número de dias após a senha expirar que a conta deverá ser desabilitada.
8	*Expire*	`39378`	Uma data fixa, medida em dias após 01/01/1970, em que a senha irá expirar.

Criptografia da Senha

A senha criptografada é dividida em três partes, separadas pelo caractere $. A figura a seguir mostra uma visão geral dessas três partes:



A tabela a seguir descreve cada um dos campos:

Campo #	Nome	Exemplo	Descrição
1	*Type*	`6`	O tipo de criptografia usada: 1 = MD5, 2=Blowfish, 5=SHA-256, 6=SHA-512
1	*Salt*	`pii2cuKVIBrbTtAG`	Salt é uma técnica usada para aumentar a segurança do *hash*. Será detalhado a seguir.
1	*Hash*	`Dkm.Xuiz...x0p03cE0`	*Hash* da senha resultante do algoritmo de criptografia.

Muito provavelmente, as senhas do seu sistema devem usar o SHA-512 (*Type* 6), que é o algoritmo recomendado atualmente para senhas seguras. O SHA-512, assim como os outros algoritmos, é conhecido como função hash criptográfica. Estes algoritmos têm como entrada uma string ou texto e geram como saída uma assinatura (*hash*) que é (quase) única. Na verdade, a principal característica desses algoritmos é que é praticamente impossível reverter o processo, ou seja, dada uma assinatura, não é possível gerar o texto original.

No caso do Linux, quando você define a sua senha pela primeira vez, o sistema gera o *hash* da sua senha e o salva no `/etc/shadow`. Estas são as senhas criptografadas que você viu na saída do comando anterior. Conforme mencionado, a partir desses *hashes*, não é possível (em tempo útil) descobrir a senha original. Então, quando você faz login, como que o Linux sabe se a sua senha está correta ou não? Basicamente, ele gera o *hash* da senha que você acabou de digitar e o compara com o armazenado no `/etc/shadow`. Se for igual, você digitou a senha correta. Em outras palavras, a sua senha nunca é "descriptografada", pois isso é impossível (acredita-se).

Portanto, dado um *hash*, a única forma de se "descobrir" a senha original (se você for um atacante) é testando todas as possibilidades. Por exemplo, podemos começar com a senha "`aaa`", gerar o *hash* dessa string compará-lo com o armazenado no `/etc/shadow`. Se for igual, você descobriu a senha. Caso contrário, vamos tentar outra possibilidade (e.g., `aab`). Como a quantidade possível de senhas (com 8 caracteres) é um número absurdamente grande (72^8 = $7{,}2*10^{14}$), essa senha não seria descoberta a tempo de ser útil, pois poderia demorar séculos para descobri-la.

Mas calma! Ainda tem mais! Para evitar que o atacante coloque o *hash* no Google e encontre uma possível palavra (senha) para o *hash*, o Linux usa uma técnica conhecida como *salt*. Basicamente, se a sua senha for `lala` e o *salt* for `HL3` (por exemplo), o Linux irá gerar o *hash* não só da sua senha, mas dela concatenada com o *salt*, ou seja, `lalaHL3`. O *salt* original é salvo junto com a sua senha, como mostrado anteriormente.

Chega de conceitos. Vamos tentar gerar uma senha criptografada? O Linux tem um comando para isso chamado `mkpasswd`. Dado uma senha e um *salt*, ele é capaz de gerar uma senha criptografada:

```
$ echo -n "ncc1701" | mkpasswd --stdin --method=sha-512 --salt="pii2cuKVIBrbTtAG"
```

```
$6$pii2cuKVIBrbTtAG$ucO7i3p.8fv0z7ZBLr.WfHZiuVqwg.A.TNrR8qa6BNEF4Ecfd9abAsx4bAmoZhZssHeRwx04VcfzJG7
```

Se o seu Laptop/PC não tiver o comando `mkpasswd`, instale o pacote `whois`: `sudo apt install whois`

Desafio: Vamos tentar gerar a sua senha criptografada! Antes de mais nada, execute o comando abaixo para evitar que o `bash` salve o histórico da sua sessão para a sua senha não ficar aparecendo no seu histórico de comandos:

```
$ export HISTFILE=/dev/null
```

Veja a sua senha criptografada no `/etc/shadow` para descobrir o *salt* que o Linux usou para salvar a sua senha. Em seguida, execute o comando `mkpasswd` acima colocando sua senha (real) depois do `echo` (entre aspas) e mudando o *salt* no final do comando. A senha criptografada gerada deve ser igual à do `/etc/shadow`.

3. Grupos

Assim como as contas de usuários, os grupos no Linux são definidos em um arquivo: `/etc/group`

```
$ cat /etc/group
```

Como você poder ver, são diversos grupos. Vamos mostrar apenas os 3 primeiros:

```
$ head -3 /etc/group
```

```
root:x:0:
daemon:x:1:
bin:x:2:
```

Olhe também o grupo que o usuário que você criou pertence:

```
$ cat /etc/group | grep '^masp:'
```

```
masp:x:1001:
```

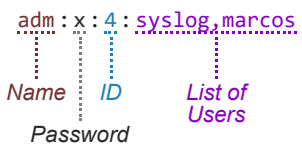
No Linux, por padrão, ao criar um usuário é criado também um grupo com o mesmo nome. Se os usuários pertencessem a um mesmo grupo (e.g., `normal-users`), dependendo das permissões, os usuários poderiam acessar os arquivos dos outros usuários, o que não é recomendado.

Por fim, alguns grupos possuem uma lista de usuários no final:

```
$ cat /etc/group | grep '^adm:'
```

```
adm:x:4:syslog,marcos
```

Cada linha do arquivo `/etc/group` possui 4 campos separados por `:` (dois pontos). A figura a seguir mostra um exemplo de uma linha:



A tabela a seguir descreve cada um dos campos:

Campo #	Nome	Exemplo	Descrição
1	*Name*	`adm`	Nome do grupo.
2	*Password*	`x`	Normalmente, uma senha não é usada, por isso o `x`. Mas esse campo pode ser usado para armazenar uma senha para o grupo.
1	*ID*	`4`	ID do grupo, também conhecido como GID (*group id*).
1	*List*	`syslog,marcos`	Lista de usuários do grupo. Normalmente, este campo é vazio, pois o grupo principal de um usuário é definido no `/etc/passwd`. Esse campo só é necessário

quando um usuário precisa pertencer a vários grupos (além do seu principal).

Para adicionar um novo grupo, use o comando `addgroup`:

```
$ sudo addgroup <GROUP_NAME>              # Substitua <GROUP_NAME> pelo NOME do novo grupo (sem <>)
```

```
Adding group `devtitans' (GID 1002) ...
Done.
```

Veja se o seu novo grupo foi criado no `/etc/group`:

```
$ cat /etc/group | grep '^devtitans:'
```

```
devtitans:x:1002:
```

Para adicionar um usuário em um grupo, use o comando `usermod`:

```
$ sudo usermod -a -G devtitans masp
```

Veja se o seu usuário foi realmente adicionado ao novo grupo:

```
$ cat /etc/group | grep '^devtitans:'
```

```
devtitans:x:1002:masp
```

Por fim, você pode ver todos os grupos que um usuário pertence através do comando `groups`:

```
$ groups masp
```

```
masp : masp devtitans
```

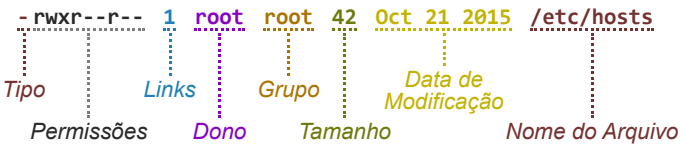
4. Permissões de Arquivos

Uma vez que você efetuou logon com seu usuário e senha, o restante da segurança é feito através de controle de acesso via permissões. Cada usuário logado possui as suas credenciais, que podem ser vistas através do comando `id`:

```
$ id
```

```
uid=1000(marcos) gid=1000(marcos) groups=1000(marcos),4(adm),20(dialout),24(cdrom),25(floppy),
27(sudo),29(audio),30(dip),44(video),46(plugdev),117(netdev)
```

Tais credenciais são verificadas pelo kernel sempre que o usuário acessa um arquivo (dentre outras ações). No primeiro laboratório, detalhamos a saída do comando `ls -l` através da figura abaixo:



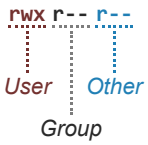
O controle de acesso ao arquivo é feito comparando os campos *tipo*, *permissões*, *dono* e *grupo* com as credenciais do usuário tentando acessar o arquivo.

Sobre o primeiro campo da saída do `ls`, o *tipo* do arquivo, ele pode assumir os seguintes valores:

Valor	Tipo de Arquivo
`-`	Arquivo Normal
`d`	Diretório
`l`	Link
`c, b, p, s`	Arquivo especial de caractere, bloco, pipe e socket.

Campo de Permissões

Sobre o campo de *permissões* do arquivo, este é dividido em três partes, conforme a figura abaixo:



Os três primeiros caracteres mostram as permissões para o usuário que é o *dono* do arquivo. O primeiro caractere diz se o usuário pode ler (`r`) ou não o arquivo. O segundo caractere diz se o usuário pode escrever (`w`) no arquivo e, por fim, o terceiro caractere diz se o usuário pode executar (`x`) o arquivo.

Os próximos três caracteres indicam essas mesmas três permissões para os usuários que fazem parte do mesmo *grupo* do arquivo. No caso do exemplo da figura acima, os usuários pertencentes ao grupo do arquivo (`root`) só podem ler o arquivo (`r`).

Por fim, os três últimos caracteres indicam as permissões para todos os outros usuários (*other*) que não são o dono do arquivo nem pertencem ao grupo dele.

Mudando as Permissões de Arquivos

O comando usado para mudar as permissões de um arquivo é o `chmod`. Para vermos o seu funcionamento, vamos criar um *script bash* que poderemos executar depois:

```
$ echo '#!/bin/bash'           >  HelloBash.sh
$ echo 'echo "Hello Bash ..."' >> HelloBash.sh
```

Veja o conteúdo do arquivo:

```
$ cat HelloBash.sh
```

Veja as permissões padrões que o arquivo foi criado:

```
$ ls -l HelloBash.sh
```

```
-rw-r--r-- 1 marcos marcos 34 May  9 19:48 HelloBash.sh
```

Como você pode ver, o dono do arquivo não pode executá-lo. Pode apenas ler e escrever nele. Vamos tentar executar assim mesmo e ver o que acontece:

```
$ ./HelloBash.sh
```

```
-bash: ./HelloBash.sh: Permission denied
```

Finalmente, vamos mudar as permissões do arquivo usando o `chmod`: Para adicionar a permissão de execução do arquivo para o usuário dono do mesmo, use a seguinte sintaxe:

```
$ chmod u+x HelloBash.sh
```

O comando acima adiciona (`+`) a permissão de execução (`x`) para o usuário dono do arquivo (`u`). Vamos listar o arquivo para ver suas permissões novamente:

```
$ ls -l HelloBash.sh
```

```
-rwxr--r-- 1 marcos marcos 34 May  9 19:48 HelloBash.sh
```

Note como agora o usuário pode executar. Vamos testar:

```
$ ./HelloBash.sh
```

```
Hello Bash ...
```

Obviamente, se queremos mudar várias permissões de uma só vez, o comando `chmod` pode ficar bem confuso. Para isso, este comando permite setar as permissões através de bits, normalmente representados por números octais. A tabela a seguir mostra

cada uma das oito possibilidades de permissões:

Valor Octal	Valor Binário	Valor Texto	Descrição da Permissão
0	000	---	Nenhuma permissão
1	001	--x	Somente execução
2	010	-w-	Somente escrita
3	011	-wx	Escrita e execução
4	100	r--	Somente leitura
5	101	r-x	Leitura e execução
6	110	rw-	Leitura e escrita
7	111	rwx	Leitura, escrita e execução

Para mudar as permissões para o dono, grupo e outros, usamos três números:

```
$ chmod 700 HelloBash.sh
$ ls -l HelloBash.sh
```

```
-rwx------ 1 marcos marcos 34 May  9 19:48 HelloBash.sh
```

O comando acima seta as permissões de leitura, gravação e execução para o usuário (`7`), e retira todas as permissões para o grupo (`0`) e para os outros (`0`).

```
$ chmod 755 HelloBash.sh
$ ls -l HelloBash.sh
```

```
-rwxr-xr-x 1 marcos marcos 34 May  9 19:48 HelloBash.sh
```

O comando acima seta as permissões de leitura, gravação e execução para o usuário (`7`), e seta as permissões de leitura e execução (`5`) para o grupo e para os outros. Ou seja, todos podem ler e executar o arquivo, mas só o dono pode alterar (escrever) o mesmo.

5. Mudando o Dono e o Grupo de um Arquivo

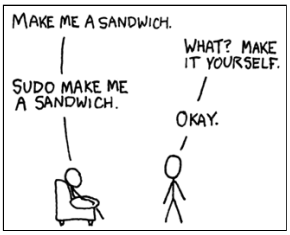
Para mudar o dono de um arquivo, usamos o comando `chown`. Já para mudar o grupo de um arquivo, usamos o comando `chgrp`. Ambos os comandos não podem ser executados por usuários normais, apenas pelo `root`. Como a sintaxe dos dois é parecida, testaremos apenas a mudança do grupo de um arquivo.

Execute o comando abaixo para mudar o grupo do arquivo criado anteriormente:

```
$ sudo chgrp devtitans HelloBash.sh
$ ls -l HelloBash.sh
```

```
-rwxr-xr-x 1 marcos devtitans 34 May  9 19:48 HelloBash.sh
```

6. Executando Comandos como Root



Várias vezes neste laboratório atual, e algumas nos anteriores, foi necessário executar o comando `sudo` para executar um comando como `root`. Finalmente, vamos falar um pouco dele!

Antigamente, a forma mais usada para "*virar root*" era executando o comando `su`. Esse comando existe até hoje. O problema deste comando é que ele pede a senha do `root`. Portanto, para que várias pessoas pudessem executar um comando como root, seria necessário distribuir a senha dele para todos. Os usuários, além de decorar a senha deles, precisavam decorar a senha do *root*.

A principal vantagem do `sudo` é que ele pede a senha do próprio usuário, e não a senha do *root*.
Além disso, o `sudo` deixa registrado nos logs do sistema quem executou o que, permitindo um controle e um registro mais detalhado de quem fez mudanças no sistema.

O `sudo` é configurado através do arquivo `/etc/sudoers`:

```
$ sudo cat /etc/sudoers
```

Se você analisar bem o arquivo, verá que uma das linhas configura que os usuários do grupo `sudo` podem executar o comando `sudo`. Portanto, o controle de quem pode "*virar root*" é feito basicamente através desse grupo. Para permitir que um determinado usuário possa executar comandos como `root`, basta colocá-lo neste grupo.

The fruit of your own hard work is the sweetest. – Deepika Padukone

π