Assignment 5 - Data Security

Pedro Augusto Ennes de Martino Camargo - PG59791

Question 1

Este assignment foi feito em C. As três funções (protect, update e verify) foram feitas em três ficheiros separados, dentro da pasta ***/code,*** *cada uma exerce a função pedida no enunciado.*
Para facilitar a execução da questão, foi criado um script em bash para a automatização da compilação e criação do ***file_shadow***

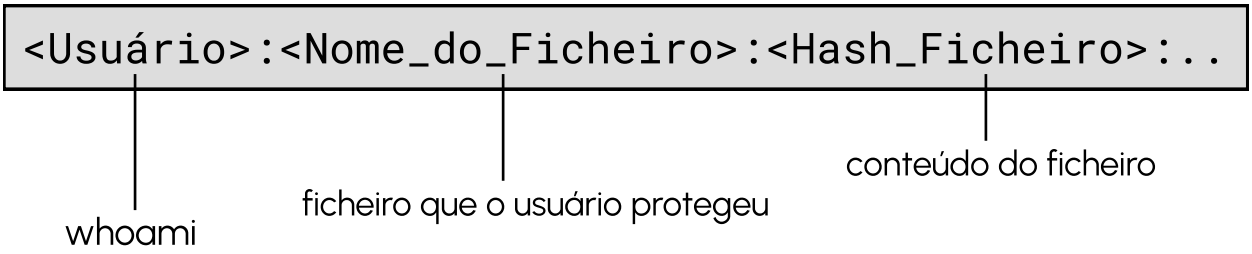
Estrutura

A forma como o file_shadow foi dividido foi pensada de forma a possibilitar a presença de diferentes tipos de usuários para vários ficheiros distintos.

Ou seja, o nome do meu usuário padrão é **mutante,** ele possuirá uma linha do file_shadow dedicada somente para seus ficheiros protegidos.

Caso outro usuário esteja presente num sistema (i.e. **root**), uma nova linha será criada para proteção de seus ficheiros, impossibilitando o conflito entre 2 usuários distintos

O **file_shadow** está estruturado da seguinte forma:



Aqui está um exemplo da estrutura na prática:

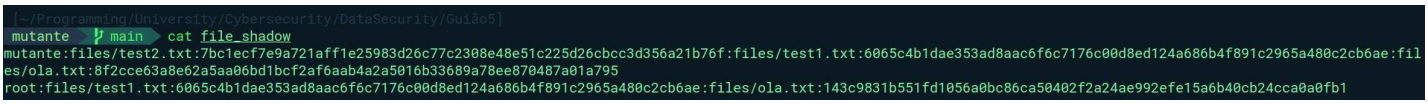


Imagem 1: Exemplo do file_shadow

Prática

Alguns ficheiros foram adicionados dentro da pasta files para o fim de servir como exemplo na utilização das funções de integridade de ficheiros.

1 - **Setup**

Para o setup, basta executar o seguinte comando no terminal:

```
[~/Programming/University/Cybersecurity/DataSecurity/Guião5]
mutante  main -  sudo ./setup.sh
[sudo] senha para mutante:
Creating file_shadow...
file_shadow created successfully.
Compiling source code...
Compilation completed.
```

Imagem 2: Compilação e criação do file_shadow

2 - **Protect**

O comando ./protect <filename> tem o seguinte comportamento:

- Lê o usuário **whoami**, caso o ficheiro que ele deseja proteger não esteja no file_shadow, uma nova linha será criada com o nome do usuário, o nome do ficheiro e seu conteúdo.
- Caso o ficheiro já esteja protegido, nada irá acontecer

```
[~/Programming/University/Cybersecurity/DataSecurity/Guião5]
mutante  main -  ./protect files/ola.txt
Appended new line for user: mutante

[~/Programming/University/Cybersecurity/DataSecurity/Guião5]
mutante  main -  cat file_shadow
mutante:files/ola.txt:8f2cce63a8e62a5aa06bd1bcf2af6aab4a2a5016b33689a78ee870487a01a795
```

Imagem 3: Proteção do ficheiro ola.txt

```
[~/Programming/University/Cybersecurity/DataSecurity/Guião5]
mutante  main -  sudo ./protect files/test1.txt
Appended new line for user: root

[~/Programming/University/Cybersecurity/DataSecurity/Guião5]
mutante  main -  cat file_shadow
mutante:files/ola.txt:8f2cce63a8e62a5aa06bd1bcf2af6aab4a2a5016b33689a78ee870487a01a795
root:files/test1.txt:6065c4b1dae353ad8aac6f6c7176c00d8ed124a686b4f891c2965a480c2cb6ae
```

Imagem 4: Caso com outro usuário

3 - **Verify**

O comando ./verify <filename> tem o seguinte comportamento:

- Lê o usuário **whoami**, caso o ficheiro que ele deseja verificar não esteja no file_shadow, nada irá acontecer
- Caso o ficheiro e o usuário existam, irá checar o hash do conteúdo presente no ficheiro e o hash presente no file_shadow.
- De acordo com o resultado do ponto anterior, uma resposta é retornada ao invocador da função

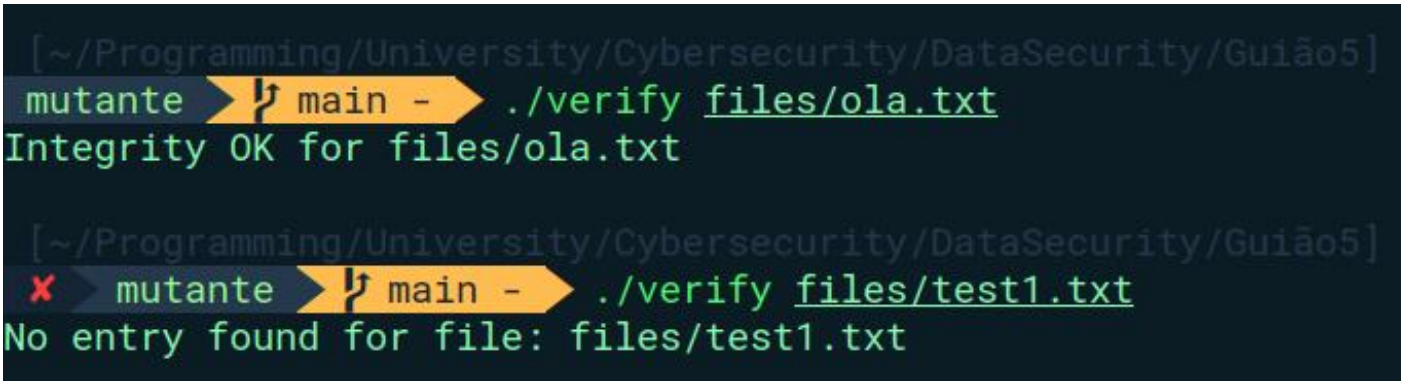


Imagem 5: Verificação de integridade

Abaixo vai um exemplo caso o ficheiro **ola.txt** sofra alguma alteração não controlada pelo utilizador (sem utilizar a função ./update)

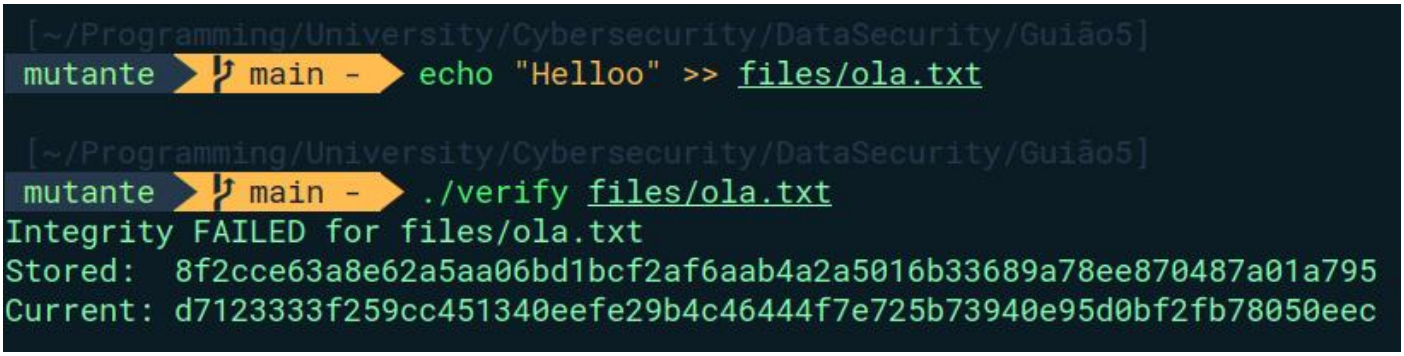


Imagem 6: Falha na verificação de integridade

4 - **Update**

O comando ./update <filename> tem o seguinte comportamento:

- Lê o usuário **whoami**, caso o ficheiro que ele deseja verificar não esteja no file_shadow, nada irá acontecer
- Caso o ficheiro e o usuário existam, irá aplicar o update do hash do ficheiro no file_shadow.

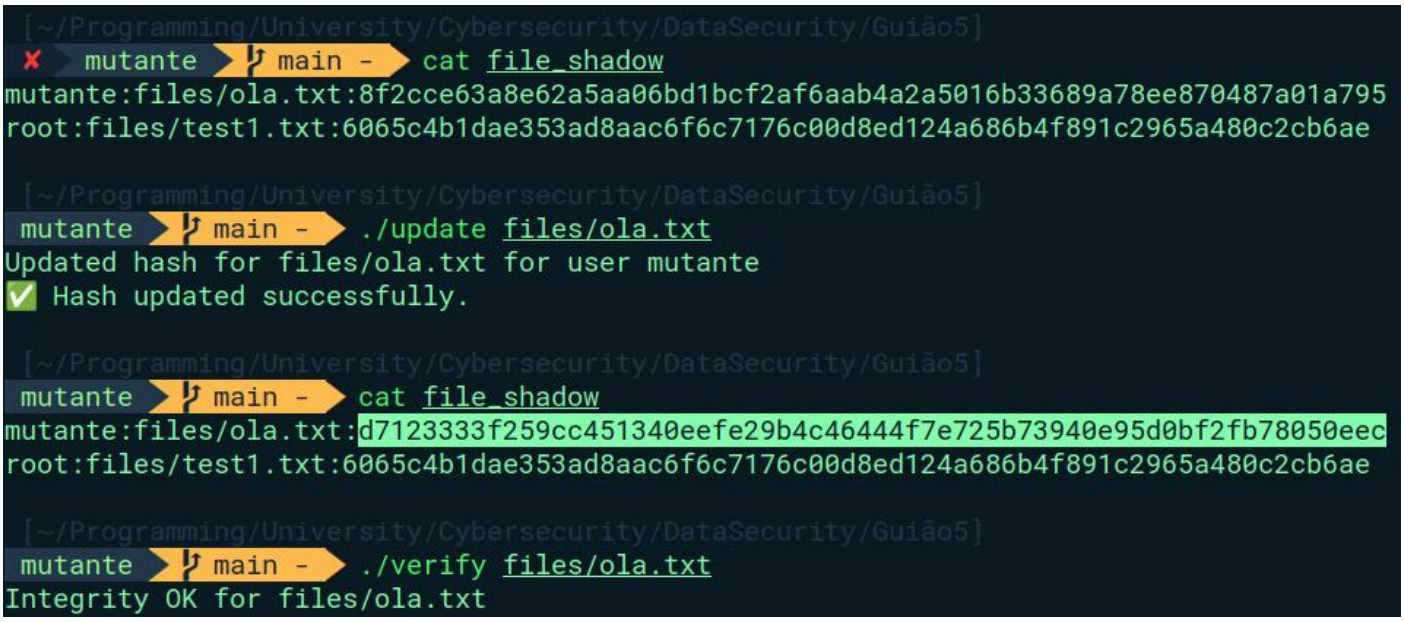


Imagem 7: Update do hash e consequente verificação

Integridade

Para garantir a integridade do ficheiro, foi utilizado o hash SHA256 junto com um salt. Como qualquer utilizador poderia modificar o próprio file_shadow sem invocar as funções, temos que adicionar um salt à função de hash, caso não houvesse salt não haveria integridade dos ficheiros.

Assim sendo, a única forma que um usuário mal intencionado poderia ter para atacar a integridade dos ficheiros era descobrir o salt do hash.

No caso deste trabalho, o salt está hard coded para **coxinha123,** não é a forma mais segura, pois é possível que um usuário possa descobrir o salt com reverse engineering dos binários **./protect, ./update** ou **./verify.**

Para evitar isso, poderíamos gerar um salt aleatório de forma a ser mais difícil encontrar o salt dentro dos binários ou passar o **salt dinamicamente** pelos argumentos da função, assim somente o usuário saberia o salt e não estaria presente dentro do binário.