SANTANDER BOOTCAMP CIBERSEGURANÇA

Explicação Código - Hashes

1. Função verificar_hashes

A função **verificar_hashes** foi criada para verificar a integridade de arquivos comparando hashes fornecidos pelo usuário (supostamente o hash real do arquivo) com o hash esperado (o valor original do hash que o arquivo deveria ter).

Entrada

- A função recebe uma lista de pares de hashes, onde cada par contém:
  - O hash calculado (fornecido pelo usuário ou obtido do arquivo).
  - O hash esperado (aquele que deveria ser o correto).
- Esses pares estão separados por vírgulas (,), e os pares em si são separados por ponto e vírgula (;).

Processamento

- Para cada par de hashes, a função compara os dois valores:
  - Se o hash calculado for igual ao hash esperado, imprime "Correto".
  - Caso contrário, imprime "Inválido".

2. Processamento do input do usuário

```
hashes_usuario = input()
```

- O código solicita que o usuário insira os pares de hashes no formato especificado:

```
"hash_calculado, hash_esperado; hash_calculado, hash_esperado"
```

- Exemplo:

```
"abc123, abc123; def456, def789"
```

```
lista_hashes = hashes_usuario.split(";")
```

- A entrada do usuário é dividida em uma lista de strings usando o ponto e vírgula como delimitador.

- Exemplo:
  - A entrada acima será convertida em:

```
lista_hashes = ["abc123, abc123", "def456, def789"]
```

3. Laço para iterar sobre os pares de hashes

```
for hash_comparacao in lista_hashes:
```

- Este laço percorre cada elemento da lista de pares de hashes (**lista_hashes**).

4. Separação de hash_calculado e hash_esperado

```
hash_calculado, hash_esperado = hash_comparacao.split(",")
```

- Cada par de hashes é dividido em dois valores separados pela vírgula.

- Exemplo:
  - Para **"abc123, abc123"**, o resultado será:

```
hash_calculado = "abc123" hash_esperado = "abc123"
```

5. Remoção de espaços extras

```
hash_calculado = hash_calculado.strip() hash_esperado = hash_esperado.strip()
```

- O método **.strip()** remove qualquer espaço em branco no início ou no final de cada string.

- Exemplo:
  - Se **hash_calculado** for **" abc123 "**, o resultado será:

```
hash_calculado = "abc123"
```

6. Comparação dos hashes

```
if hash_calculado == hash_esperado: print("Correto") else: print("Inválido")
```

Aqui é realizada a verificação:

- Se o hash calculado for **igual** ao hash esperado, significa que a integridade do arquivo está preservada. Nesse caso, imprime "Correto".
- Se os valores forem diferentes, a integridade foi comprometida, e imprime "Inválido".

7. Resultado final

A função é chamada no final do código:

```
verificar_hashes(lista_hashes)
```

- Ela recebe como entrada a lista de pares de hashes (**lista_hashes**) que foi gerada a partir da entrada do usuário e realiza as comparações.

Exemplo de Execução

Entrada do Usuário:

```
abc123, abc123; def456, def789
```

Processamento:

1. A entrada é dividida em:

```
lista_hashes = ["abc123, abc123", "def456, def789"]
```

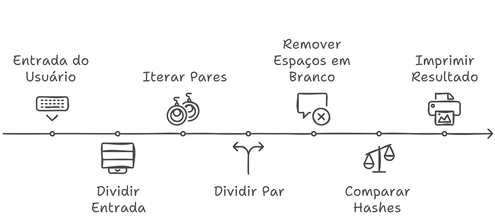
2. Iteração:

   - **Primeiro Par:**
     - **hash_calculado = "abc123"**
     - **hash_esperado = "abc123"**
     - Resultado: "Correto"

   - **Segundo Par:**
     - **hash_calculado = "def456"**
     - **hash_esperado = "def789"**
     - Resultado: "Inválido"

Saída:

```
Correto Inválido
```



Objetivo e Benefícios

Este código é útil em cenários onde você precisa verificar a integridade de arquivos, como no download de softwares, onde o hash do arquivo baixado deve corresponder ao hash fornecido pelo site oficial. Ele é simples, eficaz e pode ser facilmente adaptado para processar grandes listas de hashes ou ser integrado a sistemas mais complexos.