Estudo Dirigido: Verificação de Integridade de Arquivos com Algoritmos Hash

Parte 1 - Hash no terminal

Neste estudo dirigido vamos utilizar algoritmos hash para verificar a integridade de arquivos. Nesta atividade prática, aprenderemos a gerar e verificar hashes usando os terminais Linux e PowerShell.

Objetivos:

Compreender como os algoritmos hash garantem a integridade de arquivos.
Aprender a gerar e verificar hashes usando SHA-256.

Passo 1: Introdução

Vamos começar com uma breve explicação sobre algoritmos hash. Eles são como "impressões digitais" únicas para arquivos. Ao calcular o hash de um arquivo, obtemos uma sequência de caracteres que representa o conteúdo desse arquivo.

Passo 2: Ambiente de Trabalho

Certifique-se de que você está em um ambiente que suporta terminal Linux ou PowerShell. Vamos prosseguir com exemplos específicos.

Crie um arquivo numa pasta qualquer e abra o terminal ou powershell nesta pasta. Caso não saiba, pesquise na internet como fazer isso. 🔍

Passo 3: Geração do Hash para o Arquivo Original

Agora, vamos calcular o hash SHA-256 para um arquivo específico. Substitua "arquivo_original.txt" pelo nome do seu arquivo real.

Linux:

```
sha256sum arquivo_original.txt > hash_original.txt
```

PowerShell:

```
Get-FileHash -Algorithm SHA256 -Path .\arquivo_original.txt | Format-List > hash_original.txt
```

Este comando cria um arquivo chamado hash_original.txt, que contém o hash SHA-256 do arquivo original. A opção -Algorithm SHA256 especifica o algoritmo de hash.

Passo 4: Verificação de Integridade

Agora, vamos verificar a integridade do arquivo em relação ao hash gerado.

Linux:

```
sha256sum -c hash_original.txt
```

PowerShell:

```
Get-Content .\hash_original.txt | ForEach-Object { $_ -replace ':\s*.*', '' } | Get-FileHash -Algorithm SHA256 -Hash | Compare-Object -ReferenceObject $(Get-Content .\hash_original.txt)
```

Este comando compara o hash atual do arquivo com o hash armazenado em hash_original.txt. Se estiver tudo correto, ou seja, se os hashs forem os mesmo, não haverá mensagens de erro.

Passo 5: Simulação de Alteração e Verificação

Vamos simular uma alteração no arquivo original, adicionando ou removendo conteúdo.

Linux:

```
echo "Conteúdo modificado." >> arquivo_original.txt
```

PowerShell:

```
Add-Content -Path .\arquivo_original.txt -Value "Conteúdo modificado."
```

Agora, execute novamente o comando de verificação de integridade.

Linux:

```
sha256sum -c hash_original.txt
```

PowerShell:

```
Get-Content .\hash_original.txt | ForEach-Object { $_ -replace ':\s*.*', '' } |
Get-FileHash -Algorithm SHA256 -Hash | Compare-Object -
ReferenceObject $(Get-Content .\hash_original.txt)
```

Observe os resultados e como a verificação de integridade detectou a alteração no arquivo. 👀👩‍🏫

Passo 6: Discussão

Vamos discutir a importância da verificação de integridade em cenários do mundo real, como download de arquivos, distribuição de software e controle de versão. Que outras aplicações você acredita que podem haver para os algoritmos hash? 👩‍🏫

Passo 7: Conclusão

Com isso, aprendemos a utilizar algoritmos hash para verificar a integridade de arquivos. Eles desempenham um papel crucial na segurança de dados e garantem que os arquivos não foram corrompidos.

Continuem explorando e experimentando com algoritmos hash em diferentes contextos. Se tiverem dúvidas, não hesitem em perguntar.

Questões extras

1. Descubra quais são os outros algoritmos de hash disponíveis tanto no terminal linux ou no Powershell.
2. Execute os mesmos comandos anteriores trocando o algoritmo. Quais foram as diferenças em termos do tamanho do hash e do tempo de execução?

Parte 2 - Hash com programação

Nesta segunda parte, vocês construirão um script Python, executarão diferentes partes dele e observarão como a integridade do arquivo é afetada.

Objetivo:

Compreender como os algoritmos hash podem ser aplicados para garantir a integridade de arquivos usando Python.
Aprender a calcular e verificar hashes usando a biblioteca hashlib.
Observar as mudanças na verificação de integridade ao modificar o arquivo original.

Passo 1: Configuração do Ambiente

Certifique-se de ter o Python instalado no seu ambiente. Se necessário, faça o download em python.org e siga as instruções de instalação.

Eu considero que vocês já têm condições de executar uma aplicação python e identificar erros. Destaco, apenas, alguns pontos a relembrar:

- venv
- pip

Crie um arquivo “exemplo.txt” na mesma pasta do script python e adicione qualquer conteúdo a ele.

Passo 2: Criação do Script Python

Abra seu editor de texto ou IDE preferido e siga os passos abaixo:

```
import hashlib

def calcular_hash(arquivo):
    sha256 = hashlib.sha256()
    with open(arquivo, "rb") as f:
        # Leitura do arquivo em blocos para eficiência
        for bloco in iter(lambda: f.read(4096), b""):
            sha256.update(bloco)
    return sha256.hexdigest()

def salvar_hash(arquivo, hash_calculado):
    with open(arquivo + "_hash.txt", "w") as f:
        f.write(hash_calculado)

def verificar_integridade(arquivo, hash_salvo):
    hash_calculado = calcular_hash(arquivo)
    print(f"Hash calculado:\t {hash_calculado}")
    print(f"Hash arquivo:\t {hash_salvo}")
    return hash_calculado == hash_salvo

def main():
    arquivo_original = "exemplo.txt"
    hash_calculado = calcular_hash(arquivo_original)
    salvar_hash(arquivo_original, hash_calculado)
    print ("Hash salvo em arquivo!")

if __name__ == "__main__":
    main()
```

Explicação:

A função calcular_hash utiliza o algoritmo SHA-256 para calcular o hash do arquivo em blocos de 4096 bytes.
A função salvar_hash salva o hash calculado em um arquivo texto para referência futura.
A função verificar_integridade compara o hash atual do arquivo com o hash salvo, indicando se a integridade foi mantida.

Passo 3: Execução e Observação

Execute o script como está. Observe a mensagem indicando que o hash foi salvo no arquivo. 👀

Abra o arquivo exemplo.txt_hash.txt e verifique o hash salvo.

Passo 4: Alteração de código

Altere a função “main()” de acordo com o que está escrito abaixo. É muito importante remover todo o código anterior

```
def main():
    arquivo_original = "exemplo.txt"

    # Verificação de Integridade
    hash_salvo = open(arquivo_original + "_hash.txt").read()
    if verificar_integridade(arquivo_original, hash_salvo):
        print("A integridade do arquivo foi preservada.")
    else:
        print("Atenção! O arquivo foi modificado.")
```

O que está sendo feito no código?
O que você interpreta do resultado?

Passo 5: Alteração do arquivo

Altere o conteúdo do arquivo original manualmente, adicionando ou removendo algumas linhas.

Execute o script novamente. Observe a mensagem indicando que o arquivo foi modificado. Observe os hashes gerados. 👀

Passo 6: Restauração do arquivo

Restaure o conteúdo original e execute o script mais uma vez. 👀

Passo 7: Medindo o tempo de execução

Altere o código mais uma vez. Remova as linhas

```
if __name__ == "__main__":
    main()
```

E a adicione essa linhas

```
#esta linha vai próximo ao outro import
import timeit

#estas linhas vão para o final do arquivo
tempo = timeit.timeit(main, number=1)
print(tempo)
```

Execute e observe os resultados. 👀

Troque o valor do parâmetro number para 100 ou 1000 e observe os resultados. 👀

Passo 8: Experimentando outros algoritmos

Para desafiar vocês um pouco mais, experimentem alterar o algoritmo hash para MD5 ou SHA-1 e observem como isso afeta os resultados.

Conclusão

Parabéns! Vocês acabaram de experimentar a aplicação prática de algoritmos hash para verificar a integridade de arquivos. Espero que tenham compreendido como essa técnica é útil em diversos contextos.

Se tiverem dúvidas ou quiserem compartilhar suas observações, usem o fórum! 👩‍🎓

Questões para exercitar

1. Experimentem alterar o algoritmo hash para MD5 ou SHA-1 e observem como isso afeta os resultados de tempo.
2. Como o script detectou as modificações?
3. Qual é a importância prática disso em situações do mundo real?