Funções Hash

1st Allysson Guimarães dos Santos Silva
Bacharel em ciência da computação
Universidade Federal do Agreste de Pernambuco
Garanhuns, Brazil
allysson.guimaraes042@gmail.com

2nd Thiago Cavalcanti Silva
Bacharel em ciência da computação
Universidade Federal do Agreste de Pernambuco
Garanhuns, Brazil
tcavalcanti.tc@gmail.com

***Resumo*—As funções hash são algoritmos fundamentais na computação moderna, convertendo dados de entrada em valores hash fixos, essenciais para garantir integridade, autenticação e segurança da informação. Este artigo explora o conceito, importância e aplicações práticas das funções hash, destacando motivos para seu uso, vantagens e desvantagens. Além disso, são apresentadas soluções práticas, incluindo o cálculo de hashes de arquivos, verificação de integridade, deduplicação de dados e detecção de intrusão, com exemplos de implementação em Python. Os resultados mostram a eficácia dessas técnicas na proteção de sistemas contra ameaças, ressaltando a importância da vigilância constante e da adoção de práticas de segurança atualizadas. Integrar efetivamente as funções hash em sistemas e práticas de segurança é crucial para fortalecer a resiliência das infraestruturas digitais e proteger dados sensíveis contra ameaças cada vez mais sofisticadas.**

***Index Terms*—Funções Hash, Segurança da Informação, Integridade de Dados, Criptografia.**

I. Introdução

As funções hash desempenham um papel crucial em diversas áreas da computação e da segurança da informação. Elas são algoritmos que mapeiam dados de entrada de comprimento variável para valores de saída de comprimento fixo, geralmente em formato de sequência de caracteres. Essas funções são amplamente utilizadas em uma variedade de aplicações, incluindo criptografia, integridade de dados, autenticação e deduplicação de arquivos.

Neste trabalho, exploraremos o conceito de funções hash, sua importância e aplicações práticas. Analisaremos os motivos para usar funções hash, suas vantagens e desvantagens, além de apresentar uma solução prática com descrição e códigos, incluindo o uso de bibliotecas em Python.

Ao longo deste trabalho, examinaremos em detalhes o funcionamento das funções hash, discutindo sua relevância em contextos específicos e fornecendo exemplos práticos de implementação e uso.

II. Visão geral

As funções hash são algoritmos matemáticos fundamentais na computação moderna, usadas para transformar dados de entrada em um valor hash de saída fixo de tamanho predeterminado. Essa transformação é determinística, ou seja, a mesma entrada sempre produzirá a mesma saída.

O objetivo principal das funções hash é fornecer uma representação compacta e única dos dados de entrada, permitindo que sejam verificados, armazenados e comparados de forma eficiente. Essas funções são amplamente aplicadas em diversas áreas da ciência da computação, incluindo segurança da informação, integridade de dados, criptografia, autenticação e otimização de desempenho.

Uma característica importante das funções hash é a propriedade de dispersão. Isso significa que mesmo uma pequena alteração nos dados de entrada resultará em uma mudança significativa no valor hash de saída. Essa propriedade é essencial para garantir a integridade dos dados e a segurança das informações. Além disso, a aplicação da função a um grande conjunto de dados produzirá saídas que são igualmente distribuídas e aparentemente de modo aleatório.

Algumas aplicações que podem utilizar funções hash são:

- Autenticação de usuário: a integridade dos dados precisa ser verificada. Por exemplo, em sistemas de login, as senhas dos usuários geralmente são armazenadas como hashes para proteger sua confidencialidade.
- Autenticidade de mensagem: verifica a integridade da mensagem, garantindo que os dados recebidos estão exatamente como foram enviados (isto é, não foram modificados - inserção ou exclusão). Há algumas formas de implementar isso, podendo fornecer confidencialidade ou visando a redução de processamento.
- Integridade de dados: ao calcular o hash de um conjunto de dados, qualquer alteração nos dados resultará em um valor hash diferente, indicando que os dados foram modificados.
- Detecção de intrusão e detecção de vírus: ao armazenar o hash de cada arquivo, é possível determinar posteriormente se um arquivo foi modificado, ao verificar o cálculo de hash do arquivo. Um intruso teria que alterar o arquivo, sem modificar seu valor hash.

- Deduplicação de dados: As funções hash são empregadas para identificar e eliminar duplicatas de dados de forma eficiente. Ao calcular o hash de arquivos ou blocos de dados, é possível identificar rapidamente se dois conjuntos de dados são idênticos.

III. Motivos para usar, vantagens e desvantagens das funções hash

As funções hash são amplamente utilizadas na computação por uma série de motivos e oferecem várias vantagens significativas. No entanto, também apresentam algumas desvantagens que devem ser consideradas ao decidir sobre a aplicação.

Entre os motivos para uso estão a integridade de dados, autenticação, segurança e eficiência na busca e comparação de dados.

Dentre as suas vantagens, destacam-se a eficiência computacional e velocidade com que se calcula, mesmo para conjuntos de dados muito grandes. A representação compacta, visto que os valores hash têm um tamanho fixo que compacta a entrada, facilitando o armazenamento, transmissão e comparação. E o ponto principal, a propriedade de dispersão cuja uma pequena alteração nos dados de entrada resultará em um valor hash completamente diferente. Isso torna as funções hash robustas contra tentativas de manipulação e garante a integridade dos dados.

Embora as funções hash ofereçam uma série de vantagens, também existem algumas desvantagens a serem consideradas. Por exemplo, embora seja difícil, é teoricamente possível que dois conjuntos de dados diferentes produzam o mesmo valor hash (colisão de hash). Caso possua a propriedade de resistência à colisão, é chamada de hash forte, protegendo contra ataques onde uma mensagem poderia ser modificada e continuaria parecendo ser autêntica, pois o hash seria o mesmo. Para qualquer função de hash, é preciso haver colisões, pelo seu próprio funcionamento (tamanho de mensagem e de bloco), entretanto, deve ser computacionalmente inviável encontrar colisões para garantir a segurança.

IV. Exemplos de aplicações de funções hash

Neste capítulo, exploraremos o uso de funções hash no contexto da segurança de computadores, abordando diferentes aspectos e exemplos de aplicações práticas. Desde o cálculo de hashes de arquivos para verificação de integridade até a detecção de intrusões e vírus, examinaremos como as funções hash são empregadas para fortalecer a segurança dos sistemas.

A. Cálculo do hash de um arquivo

```
def calcular_hash
(arquivo, algoritmo='sha256'):

    hasher = hashlib.new(algoritmo)
    with open(arquivo, 'rb') as f:

        while True:
            bloco = f.read(4096)
            if not bloco:
                break
            hasher.update(bloco)

    return hasher.hexdigest()
```

A função python “calcular_hash”, é utilizada para calcular o hash de um arquivo. O código em questão define uma função, que recebe dois parâmetros:

- arquivo: Uma string que representa o caminho do arquivo para o qual o hash será calculado.

- algoritmo (opcional): Uma string que especifica o algoritmo de hash a ser utilizado. Por padrão, é definido como ’sha256’, que se refere ao algoritmo de hash SHA-256. No entanto, outros algoritmos de hash podem ser especificados, como SHA-1, MD5, etc.

A função começa criando um objeto hash usando hashlib.new(algoritmo). A biblioteca hashlib é utilizada para fornecer uma interface para vários algoritmos de hash seguros. Em seguida, o arquivo é aberto em modo de leitura binária (’rb’). O hash é calculado iterando sobre o arquivo em blocos de 4096 bytes, atualizando o objeto hash com cada bloco usando hasher.update(bloco).

Após ler todo o arquivo e calcular o hash, o valor hexadecimal do hash é retornado utilizando hasher.hexdigest().

A principal biblioteca utilizada foi a “hashlib”. Esta biblioteca é essencial para calcular hashes em Python. Ela fornece uma interface uniforme para vários algoritmos de hash, incluindo os populares SHA-1, SHA-256, MD5, etc. Neste código, ela é usada para criar o objeto de hash e calcular o hash do arquivo.

B. Verificar integridade de arquivos

```
def verificar_integridade
(arquivo, hash_esperado, algoritmo='sha256'):

    hash_calculado
    = calcular_hash(arquivo, algoritmo)
```

```
    return hash_calculado == hash_esperado
```

A função “verificar_integridade”, é usada para verificar se a integridade de um arquivo é mantida, comparando o hash calculado com um hash esperado. Esta função é uma parte importante de muitos sistemas de segurança e é fundamental para garantir que os arquivos não tenham sido alterados ou corrompidos.

A função “verificar_integridade” recebe três parâmetros:

- arquivo: Uma string que representa o caminho do arquivo cuja integridade será verificada.
- hash_esperado: Uma string que representa o hash esperado do arquivo. Este valor é comparado com o hash calculado do arquivo.
- algoritmo (opcional): Uma string que especifica o algoritmo de hash a ser utilizado. Por padrão, é definido como ’sha256’.

A função chama internamente a função calcular_hash, que foi discutida anteriormente. Isso é feito para calcular o hash do arquivo especificado. Em seguida, o hash calculado é comparado com o hash esperado. Se os dois hashes forem idênticos, a função retorna True, indicando que a integridade do arquivo é mantida. Caso contrário, retorna False, indicando que houve uma modificação no arquivo.

A verificação da integridade de arquivos é um aspecto crítico da segurança da informação. Assegurar que os arquivos não tenham sido adulterados ou corrompidos é essencial em muitos cenários, incluindo:

- Atualizações de software: verificar a integridade dos arquivos do sistema durante as atualizações de software para garantir que nenhum arquivo essencial foi modificado ou comprometido.
- Distribuição de arquivos: garantir que os arquivos distribuídos pela rede ou por outros meios permaneçam inalterados e confiáveis.
- Backup e armazenamento em nuvem: certificar-se de que os dados armazenados em backup ou na nuvem não foram corrompidos ou modificados após o armazenamento inicial.

Ao usar funções como “verificar_integridade”, os desenvolvedores podem facilmente implementar verificações de integridade em seus sistemas, melhorando assim a segurança e a confiabilidade dos dados manipulados.

C. Deduplicação de arquivos

```
def deduplicar_arquivos(diretorio):
    hashes = {}
    duplicatas = []

    # Itera sobre os arquivos no diretório
    for raiz, _, arquivos in
    os.walk(diretorio):

        for arquivo in arquivos:
            caminho_arquivo
            = os.path.join(raiz, arquivo)

            hash_arquivo
            = calcular_hash(caminho_arquivo)

            # Verifica se o hash já existe
            if hash_arquivo in hashes:

                duplicatas.append
                ((caminho_arquivo,
                hashes[hash_arquivo]))

            else:
                hashes[hash_arquivo]
                = caminho_arquivo

    # Move os arquivos duplicados para uma
    # pasta de duplicatas
    pasta_duplicatas
    = os.path.join(diretorio, 'Duplicatas')

    if not os.path.exists(pasta_duplicatas):
        os.makedirs(pasta_duplicatas)

    for duplicata, original in duplicatas:

        novo_caminho
        = os.path.join
        (pasta_duplicatas,
        os.path.basename(duplicata))

        shutil.move(duplicata, novo_caminho)

        print(f"Arquivo
        '{os.path.basename(duplicata)}' duplicado.
        Movido para {pasta_duplicatas}.")
```

A função “deduplicar_arquivos”, é utilizada para deduplicar arquivos em um diretório com base em seus hashes. A deduplicação de arquivos é uma prática importante para economizar espaço de armazenamento e otimizar a organização de dados, removendo arquivos duplicados e mantendo apenas uma cópia de cada arquivo único.

A função deduplicar_arquivos recebe um parâmetro:

- diretorio: Uma string que representa o diretório onde a deduplicação será realizada. A função começa definindo dois dicionários vazios:

- hashes: Este dicionário é usado para mapear hashes de arquivos para seus caminhos.

- duplicatas: Esta lista é usada para armazenar os pares de caminhos de arquivos duplicados encontrados durante a iteração.

A função itera sobre todos os arquivos dentro do diretório especificado usando os.walk. Para cada arquivo, o hash é calculado utilizando a função calcular_hash previamente definida. Se o hash já existe no dicionário hashes, o arquivo é considerado uma duplicata e seus caminhos são adicionados à lista duplicatas. Caso contrário, o caminho do arquivo é mapeado para o hash no dicionário hashes.

Após iterar sobre todos os arquivos, a função cria uma pasta chamada ”Duplicatas”dentro do diretório especificado, se ela ainda não existir. Em seguida, os arquivos duplicados são movidos para essa pasta usando shutil.move, renomeando-os com seus nomes originais. Um aviso é impresso para cada arquivo duplicado movido. Foram utilizadas duas principais bibliotecas para essa função:

- os: Esta biblioteca fornece funcionalidades relacionadas ao sistema operacional, como manipulação de arquivos, caminhos de diretórios, etc. É usada aqui para iterar sobre os arquivos em um diretório, criar diretórios e manipular caminhos de arquivos.

- shutil: Esta biblioteca fornece operações de alto nível em arquivos e coleções de arquivos. É utilizada aqui para mover os arquivos duplicados para a pasta de duplicatas.

A função deduplicar_arquivos fornece uma maneira eficaz de realizar a deduplicação de arquivos em um diretório, aproveitando os hashes para identificar e mover arquivos duplicados para uma pasta específica.

D. Detecção de modificação: o arquivo pode ter sido alterado por um intruso ou por vírus.

```
def verificar_integridade_arquivos
(lista_hashes, diretorio):
    for arquivo, hash_original in
    lista_hashes.items():

        caminho_arquivo
        = os.path.join(diretorio, arquivo)

        if os.path.isfile(caminho_arquivo):
            if verificar_integridade
            (caminho_arquivo, hash_original):
                print(f"O arquivo '{arquivo}'
                permanece íntegro.")
            else:
                print(f"O arquivo '{arquivo}'
                foi modificado!")
        else:
            print(f"O arquivo '{arquivo}'
            não foi encontrado no diretório.")
```

A função “verificar_integridade_arquivos”, que é utilizada para verificar a integridade dos arquivos em relação aos hashes previamente armazenados. Esta função desempenha um papel fundamental na detecção de alterações não autorizadas nos arquivos de um sistema, o que pode indicar a presença de intrusos ou malware. A função verificar_integridade_arquivos recebe dois parâmetros:

- lista_hashes: Um dicionário contendo os hashes dos arquivos previamente armazenados, onde as chaves são os nomes dos arquivos e os valores são os hashes correspondentes.

- diretorio: Uma string representando o diretório onde os arquivos estão localizados e que será verificado.

A função itera sobre todos os itens do dicionário lista_hashes, onde cada item consiste em um nome de arquivo e seu hash correspondente. Para cada arquivo, o caminho completo do arquivo é construído utilizando o diretório fornecido e o nome do arquivo. Em seguida, verifica-se se o arquivo existe usando os.path.isfile. Se o arquivo existir, a integridade é verificada chamando a função verificar_integridade discutida anteriormente. Se a integridade for mantida, uma mensagem indicando que o arquivo permanece íntegro é impressa. Caso contrário, uma mensagem indicando que o arquivo foi modificado é exibida. Se o arquivo não existir no diretório, uma mensagem informando que o arquivo não foi encontrado é impressa. A detecção de intrusão ou vírus é crucial para manter a segurança de um sistema de computador. Ao verificar a integridade dos arquivos em relação aos hashes previamente armazenados, esta função pode identificar alterações não autorizadas nos arquivos do sistema, o que pode indicar atividades maliciosas, como intrusões de hackers ou infecções por malware.

A detecção precoce de tais alterações pode permitir que as medidas de segurança adequadas sejam tomadas para mitigar os danos e proteger o sistema contra ameaças adicionais.

V. Resultados e Discussões

A função de cálculo de hash de arquivos mostrou-se eficaz na geração de representações únicas e fixas de arquivos, permitindo a verificação de sua integridade. Esta técnica é

amplamente utilizada em sistemas de segurança para garantir que os arquivos não tenham sido modificados ou corrompidos, oferecendo uma camada adicional de proteção.

Com apenas uma função, a de cálculo do hash, foi possível utilizar em diversos contextos e aplicações.

A função de verificação da integridade de arquivos foi capaz de detectar alterações não autorizadas nos arquivos, comparando os hashes calculados com os hashes esperados. Esta abordagem é essencial para a detecção precoce de intrusões e atividades maliciosas, permitindo uma resposta rápida e eficaz para proteger os sistemas contra ameaças.

A técnica de deduplicação de arquivos mostrou-se útil para otimizar o armazenamento de dados, removendo cópias duplicadas com base em seus hashes. Esta prática não só economiza espaço de armazenamento, mas também simplifica a organização de arquivos, melhorando a eficiência operacional.

A função de detecção de intrusão ou vírus foi capaz de identificar alterações não autorizadas nos arquivos do sistema, fornecendo uma camada adicional de segurança contra ameaças externas e internas. Esta técnica é crucial para manter a integridade e segurança dos dados, garantindo a confiabilidade dos sistemas de computadores.

VI. Considerações finais

As funções hash desempenham um papel essencial na segurança de computadores, oferecendo uma série de aplicações que fortalecem a integridade, confidencialidade e disponibilidade dos dados. Ao longo deste artigo, exploramos várias maneiras pelas quais as funções hash são utilizadas para proteger sistemas de computadores contra ameaças e vulnerabilidades.

Desde o cálculo de hashes de arquivos para verificação de integridade até a deduplicação de arquivos e detecção de intrusão ou vírus, vimos como as funções hash fornecem uma base sólida para uma variedade de práticas de segurança. Através do uso de bibliotecas como hashlib em Python, desenvolvemos soluções robustas para desafios comuns de segurança da informação.

É importante ressaltar que, embora as funções hash sejam poderosas, elas não são imunes a ataques e abusos. Colisões de hash e ataques de força bruta são preocupações conhecidas que exigem vigilância constante e a implementação de contramedidas adequadas.

Além disso, é crucial manter-se atualizado com as melhores práticas de segurança e acompanhar o desenvolvimento de novas tecnologias e métodos de ataque. A segurança da informação é uma jornada contínua, e a utilização eficaz das funções hash é apenas um aspecto de um programa abrangente de segurança cibernética.

Em última análise, ao integrar efetivamente as funções hash em nossos sistemas e práticas de segurança, podemos fortalecer a resiliência de nossas infraestruturas digitais e proteger os dados sensíveis contra ameaças cada vez mais sofisticadas e persistentes.

Referências

[1] Criptografia e segurança de redes, Wiiliam Stallings, 6ª edição, p. 265-277. Acesso em: 05 fev. 2024.