Quebra de Senhas

Programação de Computadores I

Ciência da Computação

Prof. Daniel Saad Nogueira Nunes

1 Introdução

Uma maneira segura de armazenar senhas é utilizar funções de hash criptográficas sobre elas e armazenar o resultado desta função, também chamado de *hash*, na base de dados. Uma função de hash criptográfica não deve ser inversível, isto é, dado um *hash*, deve ser computacionalmente difícil recuperar a senha em claro que corresponda ao hash.

Por exemplo, a senha `123456`, ao ser aplicada a função de hash `SHA256` sobre ela, produz o hash `8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c92`.

Contudo, caso tenhamos o arquivo criptografado disponível e se as senhas utilizadas forem fracas e outras medidas de segurança não forem adotadas, podemos utilizar um arquivo de hashes pré-computadores para descobrir a senha de um usuário (quebra de senha).

O objetivo deste projeto é, dados um arquivo criptografado com os nomes dos usuários e os hashes de suas senhas, e um arquivo com os hashes pré-computados juntamente com a senha correspondente, determinar a senha dos usuários, quando possível.

2 Especificação

Para conseguir atingir os objetivos, a seguinte especificação deve ser atendida.

2.1 Arquivo criptografado

O arquivo criptografado é um arquivo texto em que cada linha possui duas informações:

- Nome do usuário: possui até 32 caracteres minúsculos.
- Senha criptografada: senha criptografada através da função SHA256, possui **exatamente** 64 símbolos hexadecimais.

Estas informações estão separadas por um espaço.

Abaixo temos um exemplo de arquivo criptografado:

```
robertofontes 8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c92
danielsaad ff7e0624efbfe4c006a29afcd7dd6f189535f5191259c321a31d5472f175f370
joaoaraujo d58d736c7a967fb5f307951932734f8b0594725faa5011dbb66a8c538e635fb6
```

2.2 Arquivo pré-computado

Cada linha do arquivo pré-computado possui as seguintes informações:

- A senha (até 32 caracteres).
- O hash SHA256 da senha.

Estas informações estão separadas por um espaço. O alfabeto das senhas possui todos os símbolos entre as posições $[33, 126]$ da tabela ASCII.

Um exemplo de arquivo pré-computado é o seguinte:

```
jordan 136c67657614311f32238751044a0a3c0294f2a521e573afa8e496992d3786ba
qwerty 65e84be33532fb784c48129675f9eff3a682b27168c0ea744b2cf58ee02337c5
chocolate 7499aced43869b27f505701e4edc737f0cc346add1240d4ba86fbfa251e0fc35
123456 8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c92
swordfish b9f195c5cc7ef6afadbfbc42892ad47d3b24c6bc94bb510c4564a90a14e8b799
cocacola c46d87f1fb4a5df6df841030ffe300e8271af74b62f2545c6c1759d18932675d
flamengo d58d736c7a967fb5f307951932734f8b0594725faa5011dbb66a8c538e635fb6
```

Uma observação importante é que o arquivo com os hashes pré-computado é **garantido** estar em ordem alfabética dos hashes das senhas consideradas fracas.

2.3 Quebra da senha

Para quebrar a senha para cada usuário, o hash do arquivo pré-computado deve ser comparado com o hash do arquivo criptografado. Caso haja correspondência, a senha do usuário foi quebrada e corresponde à senha associada ao hash do arquivo pré-computado.

2.3.1 Busca binária e alocação dinâmica

Como o arquivo pré-computado está ordenado pelos valores de hash, ele pode ser carregado inteiramente em memória e a **busca binária** pode ser efetuada. Este tipo de busca funciona quando a entrada possui uma ordem e a chave associada pode ser encontrada com um número logarítmico de passos, o que é muito mais eficiente do que varrer o arquivo sequencialmente para cada usuário. Como o arquivo pode estar em memória e ele pode ser relativamente grande (alguns gigabytes), o uso de alocação dinâmica para carregar os dados do arquivo pré-computado do disco para a memória é crucial. Uma vez que os dados estejam em memória, a busca binária pode ser realizada para cada usuário para determinar se a sua senha foi quebrada.

2.4 Relatório de quebra

Para cada usuário que teve sua senha quebrada, deverá ser impresso em um arquivo texto uma linha com as seguintes informações:

- Nome do usuário.
- Senha do usuário.

As informações devem estar na ordem do arquivo criptografado e separadas por um espaço.

Utilizando como exemplo os arquivos criptografado e pré-computado anteriores, o conteúdo do relatório de quebra seria o seguinte:

```
robertofontes 123456
joaoaraujo flamengo
```

Note que a senha do usuário `danielsaad` não conseguiu ser quebrada, pois o hash não se encontra no arquivo pré-computado.

2.5 Passagem de parâmetros por linha de comando

O programa gerado deverá receber três argumentos por linha de comando na seguinte ordem:

- Caminho do arquivo criptografado.
- Caminho do arquivo pré-computado.
- Caminho do relatório de quebra.

A execução típica do programa segue a seguinte estrutura:

```
./<nome_executavel> <caminho_arquivo_criptografado>
<caminho_arquivo_pre_computado>  <relatorio_quebra>
```

2.6 Modularização

O sistema deverá ser dividido em módulos, cada qual com uma tarefa. Estes módulos podem ser organizados internamente através de várias funções e eles correspondem aos seguintes:

- Módulo de leitura: efetua a leitura dos arquivos.
- Módulo de escrita: realiza a impressão no relatório de quebra.
- Módulo de erro: responsável por informar erros ao usuário como abertura de arquivos, alocação dinâmica. Estes erros devem causar a interrupção do programa juntamente a uma mensagem explicativa para o usuário, *e.g.* `Arquivo não pôde ser aberto` ou `Impossível alocar a memória dinamicamente`.
- Módulo de busca: busca pelo hash correspondente no arquivo de pré-processado.

Os módulos devem ser organizados em arquivos separados, com seus respectivos arquivos de cabeçalho e implementação. Módulos adicionais podem ser criados caso necessário.

2.6.1 Utilização de Registros

Aconselha-se a definição e utilização de registros para estruturar as informações dos arquivos.

2.7 Construção do sistema

Um `Makefile` deverá ser produzido para a compilação dos códigos-fontes no executável e deverá ser distribuído junto ao código.

2.8 Documentação

O código deve ser bem documentado, com presença de comentários explicando os trechos mais complexos do código. Além disso, um arquivo `README.md` deve ser providenciado com a devida identificação do autor descrevendo o projeto e instruindo como o código deve ser compilado através da ferramenta `make`.

3 Critérios de correção

Deve ser utilizada a linguagem de programação `C` para a implementação do interpretador.

Para validação da correção do algoritmo, testes automatizados serão realizados, então é **crucial** que a saída esteja conforme o especificado.

Serão descontados pontos dos códigos que não possuírem indentação.

Os códigos deverão ser documentados e obrigatoriamente devem possuir a presença de um arquivo `Makefile`.

3.1 Ambiente de Correção

Para a correção dos projetos, será utilizada uma máquina de 64-bits com sistema operacional `GNU/LINUX` e compilador `GCC 10.2.0`, logo é imprescindível que o sistema seja capaz de ser compilado e executado nesta configuração.

4 Considerações

- `GDB`, `Valgrind` e ferramentas gráficas associadas podem ajudar na depuração do código.
- Este projeto deve ser executado **individualmente**.
- Não serão avaliados códigos que não compilem ou sem a presença de um `Makefile`.
- Como a correção é automatizada, deverá ser impresso apenas o que a especificação pede. Atente-se para a formatação da saída.
- A incidência de plágio será avaliada automaticamente com nota 0 para os envolvidos. Medidas disciplinares também serão tomadas.
- O trabalho deve ser entregue dentro de uma pasta zipada com a devida identificação do aluno no prazo combinado pelo ambiente virtual de aprendizagem da disciplina.