INF101 – Introdução à Programação II
Roteiro de Prática: 22 de abril de 2021

Introdução

Vamos resolver hoje o mesmo problema da aula prática passada: administrar um arquivo de senhas *(passwords)* associadas a nomes de usuários *(user login names)*. Só que agora vamos implementar duas grandes melhorias: senhas criptografadas e a estrutura contendo o banco de dados das senhas em memória primária será o dicionário da linguagem Python. Na aula passada, mantivemos o banco de dados em sua mesma estrutura de arquivo sequencial no *buffer* em memória primária. No arquivo texto, cada linha conterá o par (*login name,* senha criptografada). Os dois componentes do par serão separados por meio do caractere tab (`'\t'` em Python). A primeira coisa que o programa fará será a leitura de todo o arquivo armazenando-o em um dicionário em que a chave será o *login name* e o valor associado à chave, a respectiva senha criptografada. Depois, o programa pedirá ao usuário entrar com um par (*login name, password*). Ato contínuo, o programa pesquisará pelo *login name* no dicionário. Se for encontrado, o programa verificará se a senha digitada pelo usuário bate com a respectiva senha armazenada no dicionário. Se as senhas se casarem, então o programa emite uma mensagem de autenticação bem sucedida. Se não se casarem, o programa emite uma mensagem de advertência de falha na autenticação. Caso o *login name* não se encontrar no dicionário, ele inserirá o par (*login name, password* criptografada) ao dicionário. Quando o usuário terminar a entrada de *login names,* o programa voltará com os dados armazenados no dicionário para o arquivo texto que ficará disponível com os dados atualizados para uso futuro.

Para a implementação da criptografia das senhas, vamos usar o módulo `hashlib` da biblioteca padrão do Python. Este módulo permite codificar um *string* usando funções *hash* extremamente sofisticadas que seguem a codificação SHA-2, conforme projetada pela NSA (National Security Agency) norte-americana. Uma boa vantagem do módulo `hashlib` é que ele implementa também métodos para converter o *byte string* produzido pela cifragem para *character string,* a saber, o método `hexdigest`, que facilita o armazenamento da senha criptografada em arquivo de texto. Além disso, não é necessário manter nenhuma chave explícita, obviamente secreta, de criptografia.

Instruções

1. Faça o *download* do esqueleto do programa que se encontra no sistema de entrega do LBI. Abra o IDLE e mude o nome do arquivo-fonte para `p11.py`. Não se esqueça de salvá-lo de tempos em tempos, porque pode ocorrer falha de energia elétrica durante a aula prática.
2. Complete os comentários obrigatórios (nome, matrícula, data e uma breve descrição sobre o que o programa faz).
3. O programa está estruturado em sete funções: `main()`, `gere_dicionario(arqnome)`, `armazene(dicSenhas, arqnome)`, `hash_password(senha)`, `autenticado(clear, hashed)`, `pesquise(dicSenhas, nome)` e `insira(dicSenhas, login, senha)`.
4. Nesta aula, você completará o código-fonte que veio pronto no esqueleto com as seguintes quatro funções ainda não implementadas: `gere_dicionario`, `armazene`, `pesquise` e `insira`.
5. A função `main()` faz o seguinte: gera o dicionário contendo os pares (login name, senha cifrada) lidos do arquivo de senhas; lê o *login name* de um usuário digitado pelo usuário e, em seguida, a sua respectiva senha. Depois ela pesquisa no dicionário de senhas se o usuário se encontra lá. Caso a pesquisa tenha sucesso, ela testa se a senha retornada pela pesquisa bate com a respectiva senha cifrada que foi dada em texto claro pela entrada do usuário. Se bater, emite uma mensagem de sucesso na autenticação do usuário. Se não bater, emite uma mensagem de advertência: o usuário não foi autenticado. Caso a pesquisa no dicionário de senhas pelo *login name* obtiver fracasso, a função `main` insere o novo *login name* e a

respectiva senha cifrada no dicionário de senhas. Se o usuário digitar um *login name* vazio, isto é, digitar Enter sem nenhum caractere antes, a função main termina a interação com o usuário e finaliza o processamento voltando o dicionário de senhas para a forma de arquivo de texto usando a função `armazene`. A `main` já está pronta no esqueleto. Favor não a modificar.

6. A função `hash_password(senha)` produz a criptografia do parâmetro `senha` que deve ser um string em texto claro. Para codificar a senha, primeiro, colocamos uma pedrinha de sal, a variável `salt`, no início da senha com o propósito de dificultar a quebra da senha cifrada por algum *hacker* malicioso. Para tanto, usamos o módulo `uuid` (Universally Unique IDentifier) do Python que fornece objetos imutáveis para gerar identificadores únicos para os sistemas operacionais e também para a internet. Em particular, a função `uuid4()` gera um número aleatório de 128 bits. A função `hash_password` já está pronta. Deixe-a como está.
7. A função `autenticado(clear, hashed)` é razoavelmente simples. Ela apenas retorna o booleano resultante da igualdade entre a senha `clear` em texto claro e a senha criptografada `hashed`. Ou seja, se a senha lida pela entrada do usuário for igual à senha armazenada no dicionário de senhas referente ao *login name*, ela retorna `True`. Caso contrário, retorna `False`. A função `autenticado` já está pronta no esqueleto. Não mexa nela.
8. A função `gere_dicionario(arqnome)` é para ser implementada por você. Ela manipula o arquivo de entrada contendo os pares (*login name,* senha criptografada) em cada linha. Os dados na linha são separados por um caractere tab. Ela cria um dicionário do zero e o povoa com os pares lidos sendo a chave o *login name* e o valor associado é a respectiva senha criptografada. Se o arquivo não existir de antemão, a função deve manipular essa possibilidade, de modo que o programa não seja abortado. A função retorna o dicionário construído.
9. A função `armazene(dicSenhas, arqnome)` também será implementada por você. Ela toma o dicionário com *login names* como chaves e as respectivas senhas criptografadas como o valor associado às chaves e o salva em um arquivo de saída cujo nome está no parâmetro `arqnome`. Cada linha do arquivo de saída conterá um *login name* e a respectiva senha cifrada separados por um tab. De novo, se o arquivo não puder ser aberto por alguma razão, trate essa possibilidade como uma exceção, não deixando que o programa termine inesperadamente. A função não precisa retornar nada.
10. Implemente a função `pesquise(dicSenhas, nome)`. Nesta prática, essa função tem uma implementação surpreendentemente simples, porque o dicionário contém a informação que precisa ser retornada de acordo com a chave `nome` (que contém o *login name* a ser pesquisado). Se a chave nome não existir no dicionário, ela retorna o caractere nulo.
11. A função de inserção a ser implementada por você, `insira(dicSenhas, login, senha)`, é muito fácil também, porque se reduz à inserção da chave `login` com o valor `senha` ao dicionário `dicSenhas`. O parâmetro `senha` contém a senha em texto claro e ela deve ser criptografada antes de ser inserida no dicionário `dicSenhas`. Ela não precisa retornar nada.
12. Teste seu programa com os dados mostrados nos dois Exemplos de Teste do Programa abaixo. Os dados digitados pelo usuário estão enfatizados em fundo amarelo. Observe que o programa termina a execução ao entrar com um *login name* vazio. Então o programa emite uma mensagem de término.
13. Se seu programa entrar em *laço infinito* ou travar por alguma razão, digite `CTRL-C` na janela do *Python Shell* para interromper a execução do programa.
14. Inspecione visualmente o arquivo `passwords.hash` produzido pelo seu programa, usando para tanto um editor de texto puro, tipo `bloco de notas` no Windows ou o `gedit` no Linux, ou ainda o próprio IDLE em qualquer dos dois sistemas operacionais. Veja se seu arquivo bate com o exemplo de teste abaixo.

☞ Não se esqueça de preencher o cabeçalho do código fonte com seus dados, a data de hoje e uma breve descrição do programa.

Após certificar-se de que seu programa esteja correto, envie o arquivo com o código fonte (`p11.py`) através do Sistema de Entrega do LBI.

Exemplo 1 de Teste do Programa

```
Início do processamento do arquivo de senhas 'passwords.txt'.

Login name: lcaa
Password: 123456
O arquivo de senhas 'passwords.txt' não existe. Está sendo criado...
O usuário 'lcaa' não existe. Está sendo criado...

Login name: 90876
Password: carvalho
O usuário '90876' não existe. Está sendo criado...

Login name: flordelis
Password: camelia
O usuário 'flordelis' não existe. Está sendo criado...

Login name: 90876
Password: carvalho
O usuário '90876' foi autenticado pelo sistema.

Login name: flordelis
Password: camélia
O usuário 'flordelis' NÃO foi autenticado pelo sistema.

Login name:

Fim do processamento do arquivo de senhas.
```

Exemplo 2 de Teste do Programa

```
Início do processamento do arquivo de senhas 'passwords.txt'.

Login name: lcaa
Password: 123456
O usuário 'lcaa' foi autenticado pelo sistema.

Login name: 90876
Password: pau-brasil
O usuário '90876' NÃO foi autenticado pelo sistema.
```

```
Login name: admin
Password:
O usuário 'admin' não existe. Está sendo criado...

Login name: guest
Password: guest
O usuário 'guest' não existe. Está sendo criado...

Login name:

Fim do processamento do arquivo de senhas.
```

Arquivo `passwords.hash` produzido pelos Exemplos de Teste*

```
lcaa    7fea15eca6a9e1a40d6d4a2345def095120b5b6d740bca231ce8e047490ae879:acccf59
b754b4c13801f638287a4a961
90876   ea176094efe3d6330bffd62b0b2eabe5fa763f39335ef6417fca6bbf0e9b6d7f:114d112
015f841d784ac877502afd02c
flordelis     0342a836a01d1daeddb36cff940afea2c7900dd6919dd08d464ec0e0eca586b1:6
0344c9e633d4379a2ef6c98297d3020
admin   f8ff9d99615e61da4eecba4557804b77b85f08529794322690bacb95df8077d4:9f90579
5ed9c4675a0ce97f68d1c42d7
guest   26d85f5e41f3462475119ccb2cf6cb53acc0c471b0b70f09300d642c24cc94dc:bafb1e0
017634f50a39ab211ed32b377
```

* As senhas criptografadas produzidas na sua plataforma certamente serão diferentes das mostradas aqui, porque a pedrinha de sal colocada no meio do código e o *hash* calculado variam de sistema para sistema e com a hora em que são gerados também.