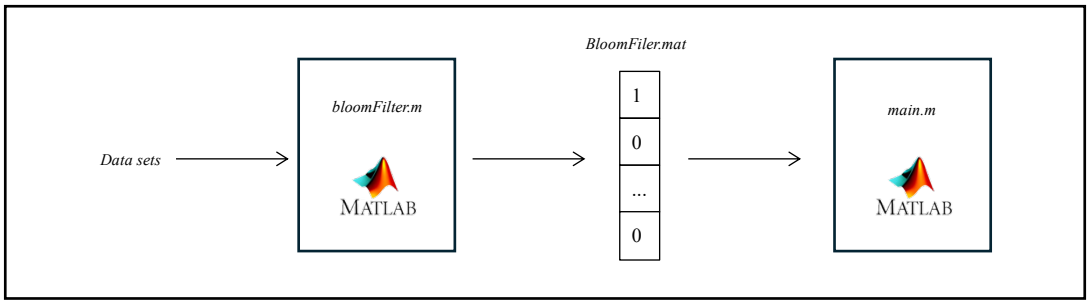
universidade de aveiro
deti departamento de eletrónica, telecomunicações e informática

Projeto MPEI - *Compromised Password Checker*

Marta Cruz - nº119572; Matilde Rodrigues - nº 119714

Resumo

O objetivo deste trabalho foi desenvolver um sistema eficiente que utilizasse 3 métodos probabilísticos na sua implementação: *Naïve Bayes*, *Bloom Filter* e *MinHash*. O projeto *Compromised Password Checker* tem como objetivo a implementação de um sistema para a identificação de palavras-passe possívelmente comprometidas. Utilizando os 3 métodos proabilísticos pedidos o sistema avalia palavras-passe introduzidas pelo utilizador, classificando-as como comprometidas ou fortes (não comprometidas). Os 3 métodos foram desenvolvidos separadamente, em 3 módulos e apenas o seu *output* foi utilizado na implementação conjunta. Abaixo apresentamos um esquema que descreve esse comportamento para o módulo *Bloom Filter* em particular.



---

1. Descrição de como correr os vários programas

Cada um dos 3 módulos implementados (*Naive Bayes*, *Bloom Filter* e *MinHash*) contém um script de testes individual. No script de teste de cada módulo escolhemos 5% das *passwords* contidas no ficheiro *common_passwords.txt* e 5% das *passwords* contidas no ficheiro *strong_passwords.txt*. Além destas, ainda adicionámos ao conjunto de *passwords* de teste algumas que não se encontram em nenhum dos *datasets*.

Assim, o *Naive Bayes* (*naiveBayes_tests*) classifica as passwords como *compromised* ou *strong* com base nas probabilidades calculadas a partir de cada caracter da *password* em questão. Já o *Bloom Filter* (*bloom_filter_tests*) deteta rapidamente se uma *password* pertence ao ficheiro *common_passwords.txt*, ao utilizar um filtro probabilístico baseado em funções de *hash*. E por fim, o *MinHash* (*MinHash_tests*) identifica se existem *passwords* similares à password em questão, sendo baseado em assinaturas produzidas a partir de todos os conjuntos de 3 caracteres pertencentes à *password* em questão (*shingles*).

Como os conjuntos de *passwords* de testes já estão criados, é só preciso correr o programa de testes para o módulo em questão e serão apresentadas no terminal as *passwords* de teste e se estão comprometidas ou não (no caso do *MinHash* vai aparece se foram encontradas *passwords* similares ou não).

1.1. Resultados obtidos

1.1.1 Naïve Bayes

No conjunto de *passwords* de teste temos 1003 *passwords*, e assim, ao executarmos o script *naive_bayes_tests* vamos obter o seguinte output (versão curta):

```
------------------ STARTING TESTS ------------------


Password: ernie  -------  Predicted: compromised   -->   True one: compromised

Password: 444444  ------- Predicted: strong   -->   True one: compromised

Password: 01021985  ------- Predicted: compromised   -->   True one:
compromised

Password: zzzzzz  ------- Predicted: strong   -->   True one: compromised

Password: oakley  ------- Predicted: compromised   -->   True one: compromised

. . .

Password: sr*+cYg4N  ------- Predicted: strong   -->   True one: strong

Password: e\dJc%A7:_f#  ------- Predicted: strong   -->   True one: strong

Password: TotallyNew#Pass  ------- Predicted: strong   -->   True one: strong

Password: Unkown@Password  ------- Predicted: strong   -->   True one: strong

Password: paris12  ------- Predicted: compromised   -->   True one: compromised


Accuracy: 0.995015

Precision: 1.000000

Recall: 0.990020

f1: 0.994985


------ TESTS COMPLETED ------
```

As primeiras *passwords* são do ficheiro *common_passwords.txt*, e as últimas são do ficheiro *strong_passwords.txt*, sendo que as últimas 3 são as *passwords* externas que foram adicionadas e que não pertencem a nenhum ficheiro. Avaliando as métricas apresentadas no final do teste, apercebemo-nos que este método probabilístico apresenta algumas falhas, sendo algumas palavras-passe marcadas falsamente comprometidas e falsamente fortes, daí não estarem todas igual a 1. Verificamos isto nomeadamente nas *passwords* que contenham apenas 1 caracter repetido múltiplas vezes. Este fenómeno dá-se devido ao algoritmo utilizado que apenas compara a probabilidade de 1 caracter de aparecer em ambos os *datasets*. Em qualquer que seja que apareça mais vezes, será a probabilidade dominante, possivelmente marcando erradamente a *password*.

1.1.2 Bloom Filter

No conjunto de *passwords* de teste temos 1003 *passwords*, e assim, ao executarmos o script *bloom_filter_tests* vamos obter o seguinte output (versão curta):

```
------------------ STARTING TESTS ------------------


Password: 23111989  ->  compromised

Password: 1002  ->  compromised

Password: hard  ->  compromised

Password: 09031988  ->  compromised

Password: 1111111111  ->  compromised

. . .

Password: jM&OM[_Pr(Go  ->  no compromised

Password: 6n5}O%!NLTAB  ->  no compromised

Password: TotallyNew#Pass  ->  no compromised

Password: Unkown@Password  ->  no compromised

Password: paris12  ->  no compromised


Accuracy: 1.000000

Precision: 1.000000

Recall: 1.000000

f1: 1.000000


------ TESTS COMPLETED ------
```

Assim, concluímos que, segundo as métricas apresentadas no final do output, o *Bloom Filter* funciona na totalidade, com os resultados pretendidos. Tal como nos módulos anteriores, a ordem das *passwords* é a mesma, e, portanto, os resultados fazem sentido: apenas as *passwords* que se encontram no ficheiro *common_passwords.txt* é que estão comprometidas (como as *passwords* externas não pertencem a nenhum ficheiro, não estão comprometidas, mas isso não quer dizer que sejam fortes).

1.1.3 MinHash

No conjunto de *passwords* de teste temos 1003 *passwords*, e assim, ao executarmos o *MinHash_tests* vamos obter o seguinte output (versão curta):

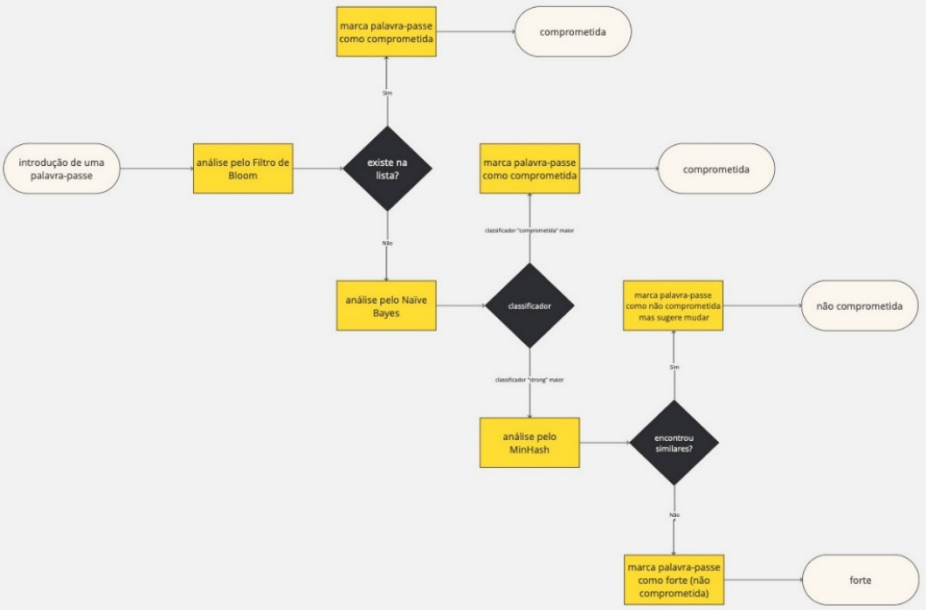
```
------------------ STARTING TESTS ------------------


Password: dragoon -> Similar passwords found

Password: bean -> Similar passwords found

Password: everett -> Similar passwords found

Password: 07061988 -> Similar passwords found

Password: bigbob -> Similar passwords found

. . .

Password: |.+~-&M@>j@[ -> No similar password found

Password: Sp<`Tq#iqbr7 -> No similar password found

Password: TotallyNew#Pass -> No similar password found

Password: Unkown@Password -> No similar password found

Password: paris12 -> Similar passwords found


------ TESTS COMPLETED ------
```

Tal como nos módulos anteriores, a ordem das *passwords* é a mesma, e, portanto, averiguamos que o *MinHash* apresenta os resultados pretendidos. Apenas as palavras que pertencem ao ficheiro *common_passwords.txt* ou, no caso das externas, sejam parecidas às encontradas nesse ficheiro, é que apresentam a mensagem *"Similar passwords found".* Nas outras não foram encontradas similaridades.

Aplicação de uso conjunto

2.1 Descrição

A aplicação conjunta desenvolvida visa simular um sistema de avaliação de comprometimentos de uma dada palavra-passe. Integra os 3 módulos desenvolvidos de forma a fornecer uma solução robusta e eficiente. Abaixo apresentamos o diagrama de fluxo do programa:



A aplicação começa por pedir ao utilizador a introdução de uma *password* pela linha de comandos. Após a sua introdução o sistema analisa se está contida no *dataset* das 10.000 *passwords* mais comuns. Se assim for, interrompe o fluxo retornando imediatamente que a *password* introduzida se encontra comprometida. Caso contrário, avança para a análise pelo Classificador de *Naïve Bayes*. Este calculará a probabilidade da *password* introduzida se encontrar comprometida tendo em conta todos os caracteres que a compõe. Ao analisar as probabilidades de cada caracter se encontrar em cada um dos *datasets* é apresentada uma decisão final através do valor do classificador. Caso o classificador *"compromised"* tome um valor maior do que o classificador *"strong"* a *password* é marcada como comprometida, sendo o fluxo do programa interrompido. Caso contrário, o sistema prossegue para a análise segundo o método probabilístico *MinHash*. Este, por sua vez, procurará na lista de *passwords* mais comuns por similares à *password* introduzida, com um *threshold* de 0.5. Caso sejam encontradas *passwords* similares o sistema marca a *password* introduzida como não comprometida, mas sugere uma mudança para algo mais único. Caso não sejam encontradas quaisquer similaridades o sistema marca a *password* como sendo forte (não comprometida).

Testes e resultados

De modo a podermos ter uma análise mais completa da aplicação de uso conjunto, decidimos testar 4 cenários (os quais estão representados no fluxograma acima).

1º teste

Neste primeiro teste vamos inserir, na linha de comandos, uma *password* que está contida no ficheiro *common_passwords.txt*. Como podemos ver na figura abaixo, e com base no fluxograma acima, o resultado é o esperado: a password está presente no ficheiro e, portanto, é "apanhada" pelo *Bloom Filter* e não passa dele.

```
Enter your password: rudolf

Bloom Filter -> Your password is compromised! Better change it now.
```

2º teste

No segundo teste vamos inserir uma password que não está no ficheiro *common_passwords.txt*, no entanto o *Naïve Bayes* vai marcá-la como comprometida tendo em conta os seus caracteres. Podemos ver este resultado na figura abaixo:

```
Enter your password: tokyo          |   P(k | strong) = 0.0109
P(k | compromised) = 0.016366       |   P(o | strong) = 0.0079
P(o | compromised) = 0.052701       |   P(t | strong) = 0.0109
P(t | compromised) = 0.039668       |   P(y | strong) = 0.0108
P(y | compromised) = 0.017477       |   Naive Bayes -> Your password is most likely compromised. Better change it!
```

3º teste

Neste terceiro teste vamos inserir uma password que passa pelo *Bloom Filter* e pelo *Naive Bayes*, no entanto, quando chega ao *MinHash*, como tem palavras-passe similares, vai ser rotulada como comprometida, como está apresentado abaixo.

```
Enter your password: password123@!

P(! | compromised) = 0.000015
P(1 | compromised) = 0.058542
P(2 | compromised) = 0.026275
P(3 | compromised) = 0.015000
P(@ | compromised) = 0.000015
P(a | compromised) = 0.074592
P(d | compromised) = 0.026050
P(o | compromised) = 0.052701
P(p | compromised) = 0.018693
P(r | compromised) = 0.055254
P(s | compromised) = 0.048092
P(w | compromised) = 0.010946
```

```
P(! | strong) = 0.0114
P(1 | strong) = 0.0081
P(2 | strong) = 0.0128
P(3 | strong) = 0.0132
P(@ | strong) = 0.0110
P(a | strong) = 0.0110
P(d | strong) = 0.0107
P(o | strong) = 0.0079
P(p | strong) = 0.0111
P(r | strong) = 0.0112
P(s | strong) = 0.0109
P(w | strong) = 0.0113
```

```
Similar passwords found:
 1. password
 2. password1
 3. password123
 4. passwords
 5. password2
 6. mypassword
 7. password12
 8. Password123
 9. password9
10. password99

MinHash -> Your password is not compromised, however consider changing it to something more unique.
```

4º teste

No último teste vamos inserir uma *password* que seja *strong*, ou seja, que passe em todos os módulos e que não seja rotulada como comprometida, tal como podemos ver no teste abaixo efetuado:

```
Enter your password: un#a%gs

P(# | compromised) = 0.000015
P(% | compromised) = 0.000015
P(a | compromised) = 0.074592
P(g | compromised) = 0.020180
P(n | compromised) = 0.050209
P(s | compromised) = 0.048092
P(u | compromised) = 0.019804
```

```
P(# | strong) = 0.0083
P(% | strong) = 0.0115
P(a | strong) = 0.0110
P(g | strong) = 0.0113
P(n | strong) = 0.0109
P(s | strong) = 0.0109
P(u | strong) = 0.0116

MinHash -> Good job, your password is not compromised and no similar passwords were found.
```

Vantagens

A aplicação de uso conjunto combina os 3 métodos (*Bloom Filter, Naive Bayes* e *MinHash*) formando assim uma solução eficiente, robusta e abrangente para o tema abordado. O *Bloom Filter* deteta rapidamente se uma *password* pertence ao ficheiro *common_passwords.txt*, reduzindo assim as procuras desnecessárias. Já o *Naïve Bayes* aplica um modelo probabilístico que permite uma classificação mais precisa, diminuindo assim quaisquer potenciais falsos positivos criados pelo *Bloom Filter*. Por fim, o *MinHash* identifica similaridades entre *passwords*, mesmo que não sejam idênticas às *passwords* comprometidas conhecidas, abordando assim casos de variações que frequentemente aparecem em ataques.

Limitações

Apesar das vantagens, este sistema possui algumas limitações. O *Bloom Filter* e o *Naïve Bayes* podem gerar falsos positivos, classificando assim as passwords incorretamente. Além disso, o *Naive Bayes* não considera fatores importantes de segurança (comprimento da password, etc), e apresenta algumas dificuldades com passwords que possuem padrões fora do comum (repetição contínua de 1 caracter), o que leva à má classificação das passwords. Por fim, o *MinHash* é bastante sensível com os parâmetros (tamanho de *shingles* e nº de funções *hash* (k)), o que pode afetar significativamente a classificação caso sejam mal configurados.