Information theory

TP de brute-force

Emmanuel BAUDVIN
Antoine PUISSANT

Enseignant : M. FILIOL

2015 - 2016

Résumé

L'objectif de ce TP est de réaliser un ou plusieurs filtres permettant de vérifier si un fichier déchiffré est un texte réel ou une suite aléatoire de caractères.

Table des matières

1 Premiers filtres **3**

2 Encodage des caractères **3**

3 Filtre par entropie **4**

Références **6**

1 Premiers filtres

Le code qui nous a été donné nous permet, de base, d'encoder ou de décoder un fichier en fonction d'une clé donnée. Dans notre cas, nous ne connaissons pas la clé permettant de déchiffrer correctement les fichiers. Ainsi, dans un premier temps, nous avons implémenté une boucle permettant de tester toutes les clés possibles pour un fichier.
Ici, la clé fait une longueur de 59 bits soit 8 octets. Pour les besoins de résolution plus rapide, nous connaissons 44 bits de la clé (5, 5 octets). Cela laisse tout de même une possibilité de 2^{15} clés à tester (32768). Il est alors impossible de vérifier « à la main » toutes les clés.
C'est dans cette optique que nous devons implémenter des filtres. Ces derniers vont nous permettre de savoir quels sont les fichiers qui sont le plus probables d'être des textes lisibles.
Nous avons alors pensé à deux grand types de filtres.

2 Encodage des caractères

Dans un premier temps, nous avons voulu travailler avec l'encodage des caractères. En effet, nous savons ici que les documents chiffrés sont des documents textes. Il est alors simple, une fois décodé de vérifier si les caractères sont imprimables ou non.
Le premier filtre de ce genre que nous avons mis en place permet de vérifier si un caractère donné est un caractère imprimable :

```
int is_printable(mot08 in){
 int result = 0;
 if(in == ' ' || (in >= 'a' && in <= 'z'))
   return result;
}
```

Cependant ce filtre ne prend pas en compte les caractères non visibles comme le retour chariot. Il nous a alors fallu mettre en place un filtre plus évolué.
Le second filtre que nous avons mis en place permet de vérifier si le caractère décodé fait partie de la table ASCII :

```
int is_ascii(mot08 in){
 if(isprint((char)(in)) != 0 || (int)(in) == 13 || (int)(in) ==
      0 || (int)(in) == 10){
   return 0;
 }else{
   return 1;
 }
}
```

Ce dernier permet de prendre en considération les tabulations, retours à la ligne, etc... Cependant, en utilisant ce filtre sur certains documents comme *marret.cry*, il n'était pas possible de trouver un seul fichier correct. En effet, il s'avère que *marret.cry* est un texte comportant des accents. Or, la table ASCII ne comprend pas les lettres avec accents.
Ainsi, nous avons choisi d'utiliser l'UTF-8 pour l'encodage de notre troisième filtre.

L'UTF-8 est un codage de caractères informatiques conçu pour coder l'ensemble des caractères du « répertoire universel de caractères codés », initialement développé par l'ISO dans la norme internationale ISO/CEI 10646, aujourd'hui totalement compatible avec le standard Unicode, en restant compatible avec la norme ASCII limitée à l'anglais de base (et quelques autres langues beaucoup moins fréquentes), mais très largement répandue depuis des décennies.

Wikipedia[1]

Ainsi, de part la définition de l'UTF-8, nous sommes quasiment certains de pouvoir retrouver un texte, peut importe la langue dans laquelle il a été écrit. Le filtre est alors le suivant :

```
int is_utf8(mot08 in){
 if (((unsigned char)(in) >= 0x00 && (unsigned char)(in) <= 0
     x7f) || ((unsigned char)(in) >= 0xa0 && (unsigned char)(in
     ) <= 0xff)){
   return 0;
 }else{
   return 1;
 }
}
```

Grâce à ce dernier filtre, il est possible de trouver la bonne clé et de déchiffrer tous les fichiers donnés.
Il faut cependant noter que pour tous les filtres mis en place ici, nous suivons la logique suivante :

- On définie la clé à utiliser.
- Grâce à l'algorithme qui nous est donné, nous déchiffrons le premier caractère du fichier avec la clé.
- Nous donnons ce caractère à l'un de nos filtre.
  - Si le filtre valide le caractère, nous passons au caractère suivant jusqu'à erreur ou fin du fichier.
  - Si l'on rencontre une erreur, on lève un flag d'erreur.
- Si aucun flag d'erreur n'a été levé, on continue le décodage du fichier.
- Si un flag d'erreur est levé, on arrête pour cette clé et on passe à la clé suivante.
- Si nous arrivons à la fin du fichier sans avoir levé de flag, alors le fichier est validé par notre filtre. Nous pouvons l'enregistrer (les lettres décodées étant stockées dans un buffer). Et nous continuons jusqu'à la fin des clés possibles.
- Une fois les clés épuisées, le programme se termine.

3 Filtre par entropie

La seconde approche de ce projet est une approche entropique. Il s'agit d'analyser le contenu de notre fichier afin de définir si ce dernier est un texte réel ou non.
Un des filtres que nous avons testé est celui cherchant le pourcentage de 0 et de 1 dans le fichier (en binaire, une fois déchiffré). Pour ne pas avoir à déchiffrer tous les 32768 fichiers en entiers, nous faisons ici une étude sur les 100 premiers caractères de ce dernier :

```
var = lettre^outblock;
buffer[n] = (int)(lettre^outblock);
```

```
for(tmp = 0; tmp < 8; tmp++){
  if(var%2==0)
  nb_zero++;
  var >>= 1;
}
analysed_bits += 8;

if(analysed_bits >= bits_to_analyse){
  if((double)((double)nb_zero/(double)analysed_bits) < 0.515){
    printf("Walla jo z");
    flag = 100;
    analysed_bits = 0;
    bits_to_analyse = 0;
    nb_zero = 0;
    break;
  }else{
    flag = 0;
    printf("Coucou\n");
    flag = 0;
    analysed_bits = 0;
    bits_to_analyse = 0;
    nb_zero = 0;
    break;
  }
}
```

Références

[1] Wikipedia. *UTF-8*. Wikipedia. 2015. URL : https://fr.wikipedia.org/wiki/UTF-8.