TDDD86 – Extralaboration #2

12 oktober 2018

I den här uppgiften får du prova på att knäcka subsetsumkrypterade lösenord. Filerna du behöver för att komma igång finns som `extralabb2.tar.gz` på kurshemsidan. Observera att filerna i arkivet inte utgör något projekt i Qt Creator, utan det hela är baserat på make och kommandoradskörningar. Stilen koden är skriven i är även lite annorlunda jämfört med övriga labbar i kursen. Se det som en nyttig övning.

Redovisning: Efter att du redovisat muntligt, gör en `git commit -m ''TDDD86 Lab EL2 redovisning''` och en `git push`. Skicka sedan ett mail till din assistent med ämnet: `[TDDD86] Lab EL2 redovisning .` Se till att filen **decrypt.cpp** är med.

Knäcka lösenord

När ett systems inloggningshanterare ställs inför ett lösenord måste en kontroll av att det lösenordet stämmer med användarens lösenord i systemets interna tabeller utföras. En naiv metod vore att lagra lösenorden i en symboltabell med användarnas namn som nycklar, men den metoden är känslig mot att någon oauktoriserad får tillgång till systemets tabell och därmed exponerarar alla lösenord. I stället använder de flesta system en säkrare metod där en symboltabell med *krypterade* lösenord för varje användare används. När en användare skriver in ett lösenord krypteras det lösenordet och kontrolleras mot det lagrade värdet. Vid överensstämmelse släpps användaren in i systemet.

För att den här lösningen på inloggningsproblemet ska fungera väl behövs en krypteringsmetod med två egenskaper: Att kryptera ett lösenord bör vara enkelt och att lista ut ett lösenord givet den krypterade versionen bör vara mycket svårt.

SUBSET SUM-kryptering

En enkel lösenordshanteringsmetod är följande: Lösenordslängden sätts till ett specifikt antal bitar, säg N. Systemet håller reda på en tabell T med N heltal som vart och ett är N bitar långt. För att kryptera ett lösenord använder systemet lösenordet för att välja en delmängd av talen i T och lägger samman dem; summan (modulo 2^N) är det krypterade lösenordet.

Följande miniexempel med 5-bitars nycklar illustrerar processen. Antag att systemets tabell innehåller följande fem 5-bitars tal:

```
0. 10110      0
1. 01101      0
2. 00101      1
3. 10001      0
4. 01011      1
```

Med den här tabellen skulle lösenordet `00101` krypteras som `10000` eftersom det säger att summan av rad två och fyra i tabellen ska användas (och `00101 + 01011 = 10000`. I praktiken skulle förstås en mycket större tabell användas.

Antag nu att du fått tillgång till systemtabellen T och att du också lyckas snappa upp användarnamn och motsvarande krypterade lösenord. Den här informationen ger dig inte omedelbar tillgång till systemet; för att knäcka ett lösenord behöver du hitta en delmängd av T som summerar till det givna lösenordet. Säkerheten hos systemet beror på svårigheten hos det här problemet (som på engelska går under namnet SUBSET SUM). Uppenbarligen måste lösenordslängden vara tillräckligt stor för att hindra dig från att bara prova alla möjligheter men samtidigt tillräckligt liten så att användarna kan komma ihåg sina lösenord. Med N bitar finns det 2^N olika delmängder, så man skulle kunna tänka sig att 40 eller 50 bitar borde räcka…

Detaljer

I stället för att använda heltal som lösenord är det brukligt att använda någon behändig översättning från det användaren skriver in till heltal. Här har vi valt att använda ett alfabet med 32 tecken (små engelska bokstäver

samt de första sex siffrorna) i lösenorden och att koda dem som arrayer av 5-bitars heltal (`unsigned char`s) med 0 för att koda "a", 1 för att koda "b" och så vidare. Följdaktligen får vi $N = 5 \cdot C$, där C är antalet bokstäver i lösenordet.

Bland de tillhandahållna filerna finns programmet `encrypt.cpp`, med tillhörande `Makefile`, som systemadministratören skulle använda för att kryptera en användares lösenord. Programmet använder `Key.h` för att utföra addition av C-siffrors heltal i bas 32. Det läser in tabellen T och använder sedan bitarna i lösenordet för att välja ord ur tabellen att lägga samman och skriver ut det krypterade lösenordet. I arkivet finns också tabellerna `easy5.txt`, `easy8.txt`, `rand5.txt` och `rand8.txt`. Den första och tredje är för nycklar med 5 `unsigned char`s (25 bitar), den andra och den fjärde är för nycklar med 8 `unsigned char`s (40 bitar). De första två har mycket struktur medan de andra två är slumpmässigt genererade.

Med `C` definierad som `8` blir resultatet följande för lösenordet `password`:

```
>./encrypt password < rand8.txt
   password  15  0 18 18 22 14 17  3   0111100000100101001010110011101000100011
 1 gobxmqkt   6 14  1 23 12 16 10 19   0011001110000011011101100100000101010011
 2 qdrvjxwz  16  3 17 21  9 23 22 25   1000000011100011010101001101111011011001
 3 joobqxtz   9 14 14  1 16 23 19 25   0100101110011100000110000101111001111001
 4 xnoixmnk  23 13 14  8 23 12 13 10   1011101101011100100010111011000110101010
10 tcixtvem  19  2  8 23 19 21  4 12   1001100010010001011110011101010010001100
13 lqtsdtca  11 16 19 18  3 19  2  0   0101110000100111001000011100110001000000
15 zlptzlfp  25 11 15 19 25 11  5 15   1100101011011111001111001010110010101111
18 gmjuvyqw   6 12  9 20 21 24 16 22   0011001100010011010010101110001000010110
20 uoqrdhwp  20 14 16 17  3  7 22 15   1010001110100001000100011001111011001111
22 ltdkzndz  11 19  3 10 25 13  3 25   0101110011000110101011001011010001111001
23 btezrznq   1 19  4 25 17 25 13 16   0000110011001001100110001110010110110000
26 bujilqno   1 20  9  8 11 16 13 14   0000110100010010100001011100000110101110
27 qgaicljl  16  6  0  8  2 11  9 11   1000000110000000100000010010110100101011
28 yyefwcld  24 24  4  5 22  2 11  3   1100011000001000010110110000100101100011
30 gnvowyjk   6 13 21 14 22 24  9 10   0011001101101010111010110110000100101010
34 aynzobxh   0 24 13 25 14  1 23  7   0000011000011011100101110000011011100111
38 lxwewfhh  11 23 22  4 22  5  7  7   0101110111101100010010110001010011100111
39 aenipbjd   0  4 13  8 15  1  9  3   0000000100011010100001111000010100100011
   vbskbezp  21  1 18 10  1  4 25 15   1010100001100100101000001001001100101111
```

Det skulle alltså vara fullt möjligt att använda `bc` eller någon annan kalkylator för att kontrollera att kolumnerna summerar till siffror motsvarande bokstäverna i det krypterade lösenordet. Tabellen `easy8.txt` förenklar saken betydligt:

```
>./encrypt password < easy8.txt
   password  15  0 18 18 22 14 17  3   0111100000100101001010110011101000100011
 1 aaaaaaac   0  0  0  0  0  0  0  2   0000000000000000000000000000000000000010
 2 aaaaaaae   0  0  0  0  0  0  0  4   0000000000000000000000000000000000000100
 3 aaaaaaai   0  0  0  0  0  0  0  8   0000000000000000000000000000000000001000
 4 aaaaaaaz   0  0  0  0  0  0  0 25   0000000000000000000000000000000000011001
10 aaaaabaa   0  0  0  0  0  1  0  0   0000000000000000000000000000010000000000
13 aaaaaiaa   0  0  0  0  0  8  0  0   0000000000000000000000000010000000000000
15 aaaabaaa   0  0  0  0  1  0  0  0   0000000000000000000000010000000000000000
18 aaaaiaaa   0  0  0  0  8  0  0  0   0000000000000000000010000000000000000000
20 aaabaaaa   0  0  0  1  0  0  0  0   0000000000000000000100000000000000000000
22 aaaeaaaa   0  0  0  4  0  0  0  0   0000000000000000010000000000000000000000
23 aaaiaaaa   0  0  0  8  0  0  0  0   0000000000000000100000000000000000000000
26 aacaaaaa   0  0  2  0  0  0  0  0   0000000000000100000000000000000000000000
27 aaeaaaaa   0  0  4  0  0  0  0  0   0000000000001000000000000000000000000000
28 aaiaaaaa   0  0  8  0  0  0  0  0   0000000000010000000000000000000000000000
30 abaaaaaa   0  1  0  0  0  0  0  0   0000000001000000000000000000000000000000
34 azaaaaaa   0 25  0  0  0  0  0  0   0000011001000000000000000000000000000000
38 iaaaaaaa   8  0  0  0  0  0  0  0   0100000000000000000000000000000000000000
39 zaaaaaaa  25  0  0  0  0  0  0  0   1100100000000000000000000000000000000000
   b0onjjbh   1 26 14 13  9  9  1  7   0000111010011100110101001010010000100111
```

En lösning

Programmet `brute.cpp` löser problemet att givet ett krypterat lösenord och en tabell hitta lösenordet; de `C` bokstäver som, när de konverterats till ett `N`-bitars tal, bestämmer delmängden som summerar till det `N`-bitars binära tal som motsvarar de `C` bokstäver som givits som indata. Lösningen består helt enkelt i att testa alla möjliga delmängder.

Med `C` definierad till 5, blir resultatet följande för lösenordet `passw`:

```
>make encrypt
>./encrypt passw < rand5.txt
exvx5
>make brute
>./brute exvx5 < rand5.txt
i0ocs
passw
```

Observera att det inte nödvändigtvis finns en unik lösning, så programmet skriver ut allihop. Notera också att den här lösningen inte går att använda för längre lösenord. För till exempel lösenord med 10 bokstäver finns det 32^{10} (över 1 biljard) möjliga delmängder.

Symboltabellslösning

Din uppgift är att skriva ett snabbare program för att knäcka lösenord, `decrypt.cpp` som är funktionellt ekvivalent med `brute.cpp`, men tillräckligt snabbt för att klara av lösenord med 10 bokstäver med rimlig tidsåtgång (säg, under en timme).

För att få till lösningen i uppgiften ovan ska du använda en symboltabell. Grundidén är att ta en delmängd S av tabellen T, beräkna all möjliga delmängdssummor som kan skapas från S, sätta in de värdena i en symboltabell och använda den symboltabellen för att kolla de möjligheterna med bara en uppslagning.

När man betraktar strategin vi just skissat dyker flera frågor upp: Hur stor bör S vara? Exakt hur ska den här en-uppslagningskontrollen fungera? Vilken ADT bör symboltabellen baseras på? Vilken algoritm skulle vara bäst? Att handskas med de här detaljerna utgör den största delen av ditt arbete med den här uppgiften.

Det är lämpligt att debugga din lösning för lösenord med 6 bokstäver, för att sedan gå vidare till lösenord med 8 respektive 10 bokstäver. Ditt mål är, naturligtvis, att kunna dekryptera godtyckligt lösenord som krypterats med `encrypt.cpp`, som till exempel:

```
>make decrypt
>./decrypt vbskbezp < rand8.txt
koaofbmx
password
xvbyofnz
1p1ngsgg
```

Den här strategin fungerar inte heller när lösenordslängden ökar, men den minskar arbetsmängden tillräckligt för att göra system som använder den här krypteringsmetoden sårbara.

Följande lösenord har krypterats med `rand8.txt`, `rand10.txt`, respektive `rand12.txt`:

```
xwtyjjin        h554tkdzti        uz1nuyric5u3
rpb4dnye        oykcetketn        xnsriqenxw5p
kdidqv4i        bkzlquxfnt        4l4dxa3sqwjx
m5wrkdge        wixxliygk1        wuupk1ol3lbq
```

Skicka in ditt program `decrypt.cpp` tillsammans med en ifylld `readme.txt` där du svarat på frågor om din implementation (mall finns i labbens arkivfil).