Concurrencia y Paralelismo

Grado en Informática 2022

Práctica 2 – MD5

En sistemas que necesitan autenticar usando passwords es frecuente que en vez de almacenarlos directamente se guarde un hash. Cuando hay que comprobar si un password es correcto, se aplica la misma función de hash y se compara con el hash almacenado. En caso de un problema de seguridad que permita acceder a los hash los passwords no quedan expuestos porque la función no es fácilmente invertible.

Si asumiendo ciertas restricciones en la generación de passwords podemos rompelos por fuerza bruta. Vamos a partir de un programa que, a partir de un hash md5, recupere el password suponiendo:

- Los passwords tienen 6 caracteres.
- Cada carácter solo puede ser una letra minúscula distinta de la ñ.

Como excluyendo la ñ hay 26 letras, esto permite 26^6 passwords distintos. En el programa hay dos funciones (pass_to_long y long_to_pass) que establecen una correspondencia entre un número entre 0 y 26^6 y un password. Para romper el hash el programa itera entre 0 y 26^6, genera el password correspondiente, calcula su hash md5, y lo compara con el hash que queremos romper.

Para probarlo, si tenemos un password podemos generar su hash de la siguiente forma:

```
$ echo -n "passwd" | md5sum
76a2173be6393254e72ffa4d6df1030a  -
```

Y romperíamos ese hash llamando al programa y pasándolo:

```
$ ./break_md5 76a2173be6393254e72ffa4d6df1030a
76a2173be6393254e72ffa4d6df1030a: passwd
```

Para compilar el programa, hay que tener instaladas las cabeceras de desarrollo de openssl (en ubuntu o debian están en el paquete libssl-dev)

Partiendo de este código se pide:

Ejercicio 1 (**Añada una barra de progreso**) El programa trabaja sin mostrar información sobre los casos que lleva probados. Crea un thread que se encargue de imprimir una barra de progreso. El thread que está haciéndo cálculos debería informar de cuantos casos lleva probados, y el thread que imprime la barra debería usar esta información para calcular que proporción de los casos totales llevamos. No debería haber esperas activas.

Para imprimir una barra que vaya creciendo se puede usar el código \r en printf para devolver el cursor al principio de una linea. De esta forma podemos sobreescribir la información de progreso.

Ejercicio 2 (**Añada una estimación del números de passwords comprobados por segundo**) Añade al final de la barra de progreso un mensaje con el número de passwords que se están comprobando en cada segundo. Este total debe irse actualizando en tiempo real con la barra, es decir, si otros procesos usan la cpu y reducen la capacidad de cálculo disponible para comprobar hashes, el dato debería actualizarse para reflejarlo.

Ejercicio 3 (**Haga el programa multithread**) Ahora mismo se usa un único thread para romper el hash. Como el programa prueba de forma exhaustiva, podríamos dividir los posibles passwords entre varios threads y acelerar el cálculo. Añade threads para realizar los cálculos. Al obtener el resultado en cualquiera de ellos deberíamos parar al resto.

Los threads no deberían usar un reparto estático de los passwords a probar, sinó que deberían tener un contador compartido con el nº del primer password que no se ha probado. Los threads usarán esa variable para decidir cual es el número de password que deben comprobar. Para reducir la contenicón por el contador se puede hacer que los threads avancen el contador por una cierta cantidad mayor que 1, y que prueben esos passwords en secuencia. Comprueba que el número de passwords comprobados por segundo es mayor que en la versión con un solo thread.

Ejercicio 4 (**Probar varias claves a la vez**) Al romper el password por fuerza bruta estamos generando posibles password, calculando su hash md5, y comparándolo con el que queremos romper. El paso que más carga implica es el cálculo del hash, por lo que si queremos romper varios passwords es más eficiente calcular el hash asociado a uno de los posibles passwords, y compararlo con todos los hashes que queremos romper.

Modifica el programa para que acepte un número arbitrario de password por línea de comandos:

```
$ ./break_md5 76a2173be6393254e72ffa4d6df1030a 35bc8cec895861697a0243c1304c7789
```

En el momento en que se rompa un hash se imprime, y en todos los threads se deja de comparar contra él.

Entrega

La fecha límite de entrega es el 6 de marzo. El código inicial está disponible a través de github classroom https://classroom.github.com/a/NROvPR1t, donde cada estudiante podrá crear un repositorio para acceder al código y subir las sucesivas implementaciones de los ejercicios. Cubra su nombre y login en el fichero authors del repositorio.

La corrección se hará sobre el contenido del repositorio al final del día 6 de marzo.