Trabajo Práctico 2: Profiling y Optimización

Burdet Rodrigo, *Padrón Nro. 93440*
rodrigoburdet@gmail.com

Romani Nazareno, *Padrón Nro. 83991*
nazareno.romani@gmail.com

Martinez Gaston Alberto, *Padrón Nro. 91383*
gaston.martinez.90@gmail.com

1er. Cuatrimestre de 2014
66.20 Organización de Computadoras
Facultad de Ingeniería, Universidad de Buenos Aires

24 de junio de 2014

1. Objetivos

Familiarizarse con las técnicas y herramientas de profiling y optimización de software, implementando y optimizando un programa que busca descubrir la contraseñas hasheadas en un archivo.

2. Resumen

Utilizaremos la herramienta **cProfile** [1], que nos permitirá medir el tiempo que el programa pasa en cada funcion particular.
Para el desarrollo, supondremos que hemos conseguido algunos archivos con contraseñas y queremos descubrir las contraseñas originales. Sabemos que el método usado para calcular el hash es **MD5** [2], de manera que la soluciòn será ir probando cada combinación posible, comparando su hash MD5 con el de la contraseña que queremos descifrar. Sabemos que el primer archivo es de un sistema donde la gente sólo usa passwords de una letra, el segundo es de passwords de dos letras, el tercero es de passwords de tres letras, y el cuarto es de passwords de una letra, pero a los que se les agrega una 'sal'de dos caracteres, cuyo valor desconocemos, antes de calcular el hash.

2.1. Profiling

El profiling permite aprender dónde el programa pasa la mayor parte de su tiempo, y cuáles son las funciones que llaman a otras mientras se ejecuta.
Esta información puede mostrar qué piezas del programa son mas lentas de lo esperado, convirtiéndolas en candidatas para su reescritura en la etapa de optimización.
También puede ayudarnos a descubrir cuales funciones son llamadas más o menos veces de lo esperado, pudiendo así encontrar nuevos bugs (aunque el descubrimiento de bugs no es el fin principal de esta etapa).

El profiler utiliza información recolectada en tiempo de ejecución, por lo que puede ser utilizado en programas demasiado grandes o complejos, donde un análisis por lectura de fuentes sería impracticable.

Como consecuencia del análisis durante la ejecución, los datos con los que se corra el programa afectaran el resultado del profiler. Es decir, distintos datos de entrada pueden provocar distintas ramas de ejecución, dando por resultado que, por ejemplo, no se llamen algunas funciones.

2.2. Herramientas de Profiling

2.2.1. cProfile

cProfile es un modulo de python que provee herramientas de profiling y provee estadísticas de un script. Básicamente, inspecciona cada función e inserta código al principio y final de cada una, para obtener información del tiempo de ejecución. Cuando ejecutamos nuestro programa normalmente, imprimira el resultado de la corrida por la salida estandar. Para poder guardarlo en un archivo, se debe redirigir la misma, de la siguiente manera:

```
python -m cProfile [programa].py [param_1,param_2,...,param_N]
                        >[programa].stats
```

En ese archivo podremos ver, entre otras cosas, el porcentaje del tiempo de ejecucion de cada función de nuestro programa. De esta forma, podremos determinar qué funciones estan llevando mucho tiempo de ejecución, y esas serán elegidas para optimizar.

3. Desarrollo

Se trata de una versión del programa desarrollada en lenguaje python. El mismo consiste en tomar como entrada el nombre de un archivo y la longitud de las contraseñas que están hasheadas en él, encontrar qué contraseñas son buscando un string que dé el mismo hash **MD5** , y devuelva los valores de las contraseñas encontradas.

4. Ejecución

Para poder ejecutar el programa, se debe ingresar el siguiente comando:

```
python tp2.py <password_file><password_len>[salt_len]
```

5. Passwords recuperadas

Las passwords recuperadas, según cada archivo , fueron las siguientes:

```
neno@neno-laptop:/media/windows$ python tp2.py passwd_1 1
Line 1: Password 'a' matches
Line 2: Password 'b' matches
Line 3: Password 'c' matches
Line 4: Password 'd' matches
Line 5: Password '0' matches
Line 6: Password '1' matches
Line 7: Password '2' matches
Line 8: Password '3' matches
Line 9: Password 'z' matches
Line 10: Password 'p' matches
Line 11: Password 'q' matches
```

Figura 1: Passwords de una letra

```
neno@neno-laptop:/media/windows$ python tp2.py passwd_2 2
Line 1: Password 'ab' matches
Line 2: Password 'cd' matches
Line 3: Password 'ef' matches
Line 4: Password 'nn' matches
Line 5: Password 'zx' matches
Line 6: Password '81' matches
Line 7: Password '19' matches
Line 8: Password '84' matches
Line 9: Password 'zz' matches
Line 10: Password 'tp' matches
Line 11: Password 'pq' matches
```

Figura 2: Passwords de dos letras

```
neno@neno-laptop:/media/windows$ python tp2.py passwd_3 3
Line 1: Password 'Iha' matches
Line 2: Password 'vet' matches
Line 3: Password 'hef' matches
Line 4: Password 'eel' matches
Line 5: Password 'ing' matches
Line 6: Password 'Som' matches
Line 7: Password 'ebo' matches
Line 8: Password 'dys' matches
Line 9: Password 'wat' matches
Line 10: Password 'chi' matches
Line 11: Password 'ngm' matches
```

Figura 3: Passwords de tres letras

```
neno@neno-laptop:/media/windows$ python tp2_opt.py passwd_4 1 2
Line 0: Password 'a' matches
Line 1: Password 'b' matches
Line 2: Password 'c' matches
Line 3: Password 'd' matches
Line 4: Password '0' matches
Line 5: Password '1' matches
Line 6: Password '2' matches
Line 7: Password '3' matches
Line 8: Password 'z' matches
Line 9: Password 'p' matches
Line 10: Password 'q' matches
```

Figura 4: Passwords de una letra mas 'sal'de 2 caracteres

6. Optimización del algoritmo y resultados

En esta sección se expondrán los resultados obtenidos de las mejoras introducidas en el código, basados en las pruebas realizadas.
Según el profiler, el programa pasaba la mayor parte del tiempo ejecutando el algoritmo **MD5**. Por tal motivo, decidimos cambiar la implementación de nuestro programa original, que en un principio realizaba un "ataque iterativo" , para que realice un "ataque por tablas" .
De esta forma, logramos reducir la cantidad de veces que se ejecuta MD5 y, por ende, mejorar notablemente la performance de nuestro programa.

6.1. Mediciones

6.2. Mediciones de solución no optimizada

```
Line 1: Password 'Iha' matches
Line 2: Password 'vet' matches
Line 3: Password 'hef' matches
Line 4: Password 'eel' matches
Line 5: Password 'ing' matches
Line 6: Password 'Som' matches
Line 7: Password 'ebo' matches
Line 8: Password 'dys' matches
Line 9: Password 'wat' matches
Line 10: Password 'chi' matches
Line 11: Password 'ngm' matches
         14747097 function calls in 59.408 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.004    0.004    0.004    0.004 hashlib.py:55(<module>)
        6    0.000    0.000    0.000    0.000 hashlib.py:94(__get_openssl_constructor)
       11    0.000    0.000    0.000    0.000 tp2.py:16(print_password)
```

```
      1    1.432    1.432   59.368   59.368 tp2.py:25(match_passwords_file)
      1    0.000    0.000   59.408   59.408 tp2.py:4(<module>)
     11   23.836    2.167   56.632    5.148 tp2.py:8(decrypt)
      1    0.036    0.036   59.404   59.404 tp2.py:82(main)
6944781   15.218    0.000   15.218    0.000 {_hashlib.openssl_md5}
      1    0.000    0.000    0.000    0.000 {_hashlib.openssl_sha1}
      1    0.000    0.000    0.000    0.000 {_hashlib.openssl_sha224}
      1    0.000    0.000    0.000    0.000 {_hashlib.openssl_sha256}
      1    0.000    0.000    0.000    0.000 {_hashlib.openssl_sha384}
      1    0.000    0.000    0.000    0.000 {_hashlib.openssl_sha512}
     95    0.000    0.000    0.000    0.000 {chr}
      6    0.000    0.000    0.000    0.000 {getattr}
      6    0.000    0.000    0.000    0.000 {globals}
      2    0.000    0.000    0.000    0.000 {len}
      1    0.000    0.000    0.000    0.000 {method 'close' of 'file' objects}
      1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' obje
6944780   17.579    0.000   17.579    0.000 {method 'hexdigest' of '_hashlib.HASH' objec
 857375    1.304    0.000    1.304    0.000 {method 'join' of 'str' objects}
     11    0.000    0.000    0.000    0.000 {method 'rstrip' of 'str' objects}
      1    0.000    0.000    0.000    0.000 {open}
      1    0.000    0.000    0.000    0.000 {range}
```

6.3. Mediciones de solución optimizada

```
Line 1: Password 'Iha' matches
Line 2: Password 'vet' matches
Line 3: Password 'hef' matches
Line 4: Password 'eel' matches
Line 5: Password 'ing' matches
Line 6: Password 'Som' matches
Line 7: Password 'ebo' matches
Line 8: Password 'dys' matches
Line 9: Password 'wat' matches
Line 10: Password 'chi' matches
Line 11: Password 'ngm' matches
         2572297 function calls in 10.871 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.003    0.003    0.004    0.004 hashlib.py:55(<module>)
        6    0.000    0.000    0.000    0.000 hashlib.py:94(__get_openssl_constructor)
       11    0.001    0.000    0.001    0.000 tp2_opt.py:15(print_password)
        1    4.315    4.315   10.749   10.749 tp2_opt.py:24(match_passwords_file)
        1    0.000    0.000   10.871   10.871 tp2_opt.py:4(<module>)
       11    1.022    0.093    1.022    0.093 tp2_opt.py:8(decrypt)
        1    0.117    0.117   10.867   10.867 tp2_opt.py:83(main)
   857376    1.853    0.000    1.853    0.000 {_hashlib.openssl_md5}
        1    0.000    0.000    0.000    0.000 {_hashlib.openssl_sha1}
        1    0.000    0.000    0.000    0.000 {_hashlib.openssl_sha224}
        1    0.000    0.000    0.000    0.000 {_hashlib.openssl_sha256}
        1    0.000    0.000    0.000    0.000 {_hashlib.openssl_sha384}
        1    0.000    0.000    0.000    0.000 {_hashlib.openssl_sha512}
       95    0.000    0.000    0.000    0.000 {chr}
        6    0.000    0.000    0.000    0.000 {getattr}
        6    0.000    0.000    0.000    0.000 {globals}
       13    0.000    0.000    0.000    0.000 {len}
        1    0.000    0.000    0.000    0.000 {method 'close' of 'file' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' obje
   857375    2.256    0.000    2.256    0.000 {method 'hexdigest' of '_hashlib.HASH' objec
   857375    1.302    0.000    1.302    0.000 {method 'join' of 'str' objects}
       11    0.000    0.000    0.000    0.000 {method 'rstrip' of 'str' objects}
        1    0.000    0.000    0.000    0.000 {open}
```

6.4. Análisis de los datos

A continuación se detallan comparaciones de tiempo y/o uso de memoria entre la versión optimizada y la versión normal, para diferentes parámetros. Se puede notar a simple vista que la versión optimizada es más rápida que la original, dado que porque se ejecuta menos veces MD5, pero a mayor costo de

memoria.

	1	2	3	4	5
Password set	93,00	8.649,00	804.357,00	74.805.201,00	6.956.883.693,00
Password set (B)	93,00	17.298,00	2.413.071,00	299.220.804,00	34.784.418.465,00
Password set (KB)	0,09	16,89	2.356,51	292.207,82	33.969.158,66
Password set (MB)	0,00	0,02	2,30	285,36	33.173,01
Password set (GB)	0,00	0,00	0,00	0,28	32,40
Hashed Passwords (B)	2.976,00	276.768,00	25.739.424,00	2.393.766.432,00	222.620.278.176,00
Hashed Passwords (KB)	2,91	270,28	25.136,16	2.337.662,53	217.402.615,41
Hashed Passwords (MB)	0,00	0,26	24,55	2.282,87	212.307,24
Hashed Passwords (GB)	0,00	0,00	0,02	2,23	207,33
Consumo Final					
B	3069	294066	28152495	2692987236	2,57405E+11
KB	3,00	287,17	27.492,67	2.629.870,35	251.371.774,06
MB	0,00	0,28	26,85	2.568,23	245.480,25
GB	0,00	0,00	0,03	2,51	239,73
Ejecuciones de MD5 (10 casos)	1,00	2,00	3,00	4,00	5,00
TP2 (Promedio)	465	43245	4021785	374026005	34784418465
TP2 (Peor caso)	930	86490	8043570	748052010	69568836930
TP2 OPT1	93	8649	804357	74805201	6956883693

Figura 5:

Ejecuciones de MD5 (Longitud 1)	1	2	3	4	5	6	7	8	9	10
TP2 (Promedio)	46,5	93	139,5	186	232,5	279	325,5	372	418,5	465
TP2 (Peor caso)	93	186	279	372	465	558	651	744	837	930
TP2 OPT1	93	93	93	93	93	93	93	93	93	93
Ejecuciones de MD5 (Longitud 2)	1	2	3	4	5	6	7	8	9	10
TP2 (Promedio)	4324,5	8649	12973,5	17298	21622,5	25947	30271,5	34596	38920,5	43245
TP2 (Peor caso)	8649	17298	25947	34596	43245	51894	60543	69192	77841	86490
TP2 OPT1	8649	8649	8649	8649	8649	8649	8649	8649	8649	8649
Ejecuciones de MD5 (Longitud 3)	1	2	3	4	5	6	7	8	9	10
TP2 (Promedio)	402178,5	804357	1206535,5	1608714	2010892,5	2413071	2815249,5	3217428	3619606,5	4021785
TP2 (Peor caso)	804357	1608714	2413071	3217428	4021785	4826142	5630499	6434856	7239213	8043570
TP2 OPT1	804357	804357	804357	804357	804357	804357	804357	804357	804357	804357

Figura 6:

7. Conclusiones

Mediante un análisis con herramientas de perfilado como cProfile, logramos poder determinar cuáles eran los causantes del bajo desempeño del programa y cuáles eran los cuellos de botella que hacían que el programa tarde un poco más. De esta forma, pudimos optimizar el programa y lograr que se ejecute de una manera màs performante respecto del original. También pudimos aprender el manejo de estas herramientas de perfilado, y que es importante tener en cuenta el entorno en el que estamos trabajando (es decir, la jerarquía de memorias, cpu, etc), si es que queremos que un programa se ejecute de forma óptima en él.

Referencias

[1] cProfile - https://docs.python.org/2/library/profile.html
[2] MD5 - http://es.wikipedia.org/wiki/MD5