1º Cuatrimestre 2013 Ejercicio Nº 4 - El código Draka	Taller de Programación I (75.42) Facultad de Ingeniería Universidad de Buenos Aires

Ejercicio N° 4 - El código Draka

1. Introducción

Nuestros agentes han finalmente recuperado el código utilizado por los *draka* para encriptar y desencriptar sus mensajes (ver Listado 1). Si bien el código es relativamente simple, no hemos podido encontrarle vulnerabilidades hasta el momento. Solo se sabe que los textos encriptados son ASCII [1] (códigos de carácter menores a 128) y que las claves solo utilizan dígitos ASCII.

Al parecer la única opción para recuperar el texto será realizar un ataque de fuerza bruta [2].

```
void ed(uint8_t *data, size_t ndata, const uint8_t *key, size_t nkey)
{
     uint8_t i = 0, j = 0, s[256];
      do { s[i] = i; } while (++i);
      do {
          j += s[i] + key[i % nkey];
          i - j && (s[i] ^= s[j], s[j] ^= s[i], s[i] ^= s[j]);
      } while (++i);
      j = 0;
      while (ndata--) {
          i++; j += s[i];
          i - j && (s[i] ^= s[j], s[j] ^= s[i], s[i] ^= s[j]);
          *data++ ^= s[(s[i] + s[j]) & 0xff];
      }
}
```

Listado 1 - Código de encripción/desencripción utilizado.

2. Esquema de solución

Las claves utilizadas son de longitud suficiente como para hacer que un ataque por fuerza bruta con una sola máquina lleve una excesiva cantidad de tiempo. Por lo tanto, se optará por una solución distribuida siguiendo un esquema cliente-servidor: el servidor se encargará de fraccionar el trabajo y los clientes de resolver cada una de estas partes y enviar los resultados al servidor.

2.1. División del trabajo

El espacio de claves a buscar será dividido en partes lo más iguales posible y se asignarán a los clientes en el orden en que se conecten. Si buscamos probar las claves con ***d*** dígitos usando ***N*** clientes, se repartirán del siguiente modo:

- **Cliente *i* (0 ≤ *i* < *N* - 1):** claves de $i \lfloor 10^d/N \rfloor$ a $(i + 1) \lfloor 10^d/N \rfloor - 1$ (inclusive).
- **Cliente *N* - 1:** claves de $(N - 1) \lfloor 10^d/N \rfloor$ a $10^d - 1$ (inclusive).

Nota: $\lfloor x \rfloor$ es la función piso [3], que devuelve el mayor entero que sea menor o igual a ***x***.

Por ejemplo, si el trabajo consiste en probar todas las claves de 6 dígitos y se divide el trabajo entre 3 clientes, tendremos la siguiente división de tareas:

- **Cliente 0:** claves de 000000 a 333332 (inclusive).
- **Cliente 1:** claves de 333333 a 666665 (inclusive).
- **Cliente 2:** claves de 666666 a 999999 (inclusive).

Cada cliente buscará desencriptar el mensaje probando todas las posibles claves dentro del rango asignado y enviando al servidor las posibles claves que se encuentren.

2.2. Protocolo

Todas las comunicaciones serán iniciadas por un cliente y todos los mensajes intercambiados consistirán de una sola línea de caracteres imprimibles, terminada por el carácter `'\n'`. A continuación se describirán los distintos mensajes que pueden ser enviados por un cliente o por el servidor.

2.2.1. Obtener un trabajo

Es el mensaje enviado por el cliente apenas se ha conectado al servidor.

Cliente: "`GET-JOB-PART`**\n**"

La respuesta del servidor dependerá de cuándo se produzca el pedido. Si es uno de los primeros ***N*** clientes, donde ***N*** es el número de clientes que se especifica por línea de comandos, responderá con una descripción de la parte de trabajo a realizar:

Servidor: "`JOB-PART` <*mensaje-encriptado*> <*num-parte*> <*num-digitos*> <*clave-inicial*> <*clave-final*>**\n**"

donde <*mensaje-encriptado*> es una representación hexadecimal del mensaje encriptado

(usando letras mayúsculas para los dígitos hexadecimales **A**-**F**), <*num-parte*> es el número de la parte de trabajo que le corresponde (equivalente al número de cliente), <*num-digitos*> es el número de dígitos que las claves que deben probarse, <*clave-inicial*> será la primer clave a probar y <*clave-final*> será la última que deberá probarse.

Si el cliente se conecta una vez que todas las partes del trabajo ya han sido repartidas, recibirá la respuesta:

Servidor: "**NO-JOB-PART\n**"

2.2.2. Enviar una posible clave

Cuando el cliente encuentre una posible clave (una que, al desencriptar el mensaje encriptado, da como resultado texto ASCII válido) debe enviarla al servidor:

Cliente: "**POSSIBLE-KEY** <*posible-clave*>"

El servidor revisará esta clave y, de ser efectivamente una posible clave, podrá suceder una de dos cosas:

- No se encontró ninguna otra posible clave, con lo que se almacenará la posible clave recibida.
- Se encontró previamente una posible clave, con lo que el servidor almacenará la existencia de una ambigüedad.

2.2.3. Enviar finalización del trabajo asignado

Cuando el cliente termine la parte del trabajo que se le haya asignado, deberá enviar un mensaje indicando eso:

Cliente: "**JOB-PART-FINISHED** <*num-parte*>"

2.3. Forma de ejecución

El servidor se ejecutará del siguiente modo:

$ ./server <*puerto*> <*archivo-encriptado*> <*num-dígitos-clave*> <*num-clientes*>

donde <*puerto*> será el puerto que el servidor escuche, <*archivo-encriptado*> la ruta al archivo binario conteniendo los datos encriptados, <*num-dígitos-clave*> el número de dígitos de la clave y <*num-clientes*> el número de clientes entre los que se dividirá el trabajo.

La ejecución del servidor deberá poder terminarse en cualquier momento introduciendo "**q\n**" por entrada standard.

Cada cliente se ejecutará del siguiente modo:

`$ ./client` *<nombre-host>*`:`*<puerto>*

donde *<nombre-host>* será el nombre del host donde está el servidor y *<puerto>* el puerto donde el servidor está escuchando.

El cliente realizará las siguientes operaciones:

- Conectarse al servidor.
- Recibir la parte del trabajo a realizar.
- Realizar la búsqueda especificada, enviando las posibles claves encontradas.
- Terminar su ejecución.

El servidor realizará por su parte las siguientes operaciones:

- Aceptar conexiones de clientes.
- Enviarles las partes de trabajo correspondientes a realizar.
- Recibir las posibles claves.
- Imprimir la situación en cuanto a las posibles claves encontradas cuando se termine su ejecución.

Haciendo más preciso este último punto, al momento de terminar la ejecución del servidor podemos estar en uno de cuatro casos distintos:

- No haberse terminado la exploración: se imprimirá **`"Not finished"`** (sin las comillas) por salida standard.
- Haberse terminado la exploración sin haber encontrado ninguna clave: se imprimirá **`"No keys found"`** por salida standard.
- Haberse terminado la exploración encontrándose una única clave: se imprimirá la posible clave por salida standard.
- Haberse terminado la exploración encontrándose múltiples claves: se imprimirá el texto **`"Multiple keys found"`** por salida standard.

2.4. Restricciones de diseño

- Debe permitirse el cierre del servidor en cualquier momento.
- Los parámetros de entrada y archivos de datos podrán considerarse como válidos.
- La longitud de las claves a probar no será mayor que 9.
- El cliente debe cerrarse ante una interrupción de la conexión.
- El servidor debe seguir escuchando normalmente en caso de que un cliente interrumpa su conexión anormalmente.

- Deben trabajarse con el protocolo TCP/IP, utilizando sockets bloqueantes.
- Debe programarse en ISO C++ 98 [4].

Referencias

1. Wikipedia. “ASCII”. http://en.wikipedia.org/wiki/ASCII
2. Wikipedia. “Brute-force attack”. http://en.wikipedia.org/wiki/Brute-force_attack
3. Wikipedia. “Funciones de parte entera”. http://es.wikipedia.org/wiki/Funciones_de_parte_entera#Funci.C3.B3n_piso.2Fsuelo.2Fparte_entera
4. ISO/IEC. “ISO/IEC 14882 - Programming languages - C++” (1st ed). http://sites.cs.queensu.ca/gradresources/stuff/cpp98.pdf