Información Técnica de la Aplicación

Grupo 13
Cátedra Internetworking
UTN – Facultad Regional La Plata
Alfonsín Nicolás 05-18414-6
Meretta Diego 05-16424-5
Pugh Ramiro 05-18599-0

1. Objetivo

Nuestro objetivo es la creación de un software capaz de averiguar el origen de un determinado hash MD5 (Message-Digest Algorithm 5), que funcione en una arquitectura cliente servidor, donde el servidor posee el alfabeto (conjunto finito y ordenado de los símbolos a utilizar) y la clave que se quiere averiguar. Los clientes recibe dentro de cada paquete a procesar el inicio y fin de la combinatoria de palabras correspondientes, el alfabeto para realizarlo y el hash que se quiere averiguar.

2. Alcance

El Alcance propuesto para el siguiente trabajo es un servidor que cargue un alfabeto desde un archivo, así como también el hash del cual se quiere averiguar su origen y recibir conexiones de múltiples clientes a los cuales entregue el alfabeto, el inicio de la combinatoria de palabras, el fin y el hash. Por su parte el cliente posee la funcionalidad de conectarse a un servidor y realizar el cálculo de la palabra a procesar y saber si corresponde a la remitida por el servidor. Al hayarse la clave, se remite al servidor una marca avisando esto, con el número de palabra correspondiente.
Una limitación tanto del cliente como del servidor es que no podrá procesar más de 2^{64} palabras.

3.Implementación

Tanto el servidor y cliente estarán implementados en C, bajo una arquitectura TCP. La elección de la capa de transporte fue a partir de saber que necesitábamos que los paquetes lleguen en orden y no se pierdan, y de ser perdidos que los reenvíe, si no llegan en orden corremos el riesgo de estar iterando de más si en el paquete anterior hubiese llegado en el orden previsto. La implementación se hará en base a sistemas UNIX y como éste maneja los sockets, por lo tanto se obliga a que las personas que quieran ejecutar los códigos sin ningún tipo de corrección previa lo hagan bajo éstos sistemas operativos. Se utilizó el compilador GCC, y se compiló por consola.
El código posee faltas de: llamadas a excepciones, chequeo de overflow, y manejo de errores.

3.1 Servidor

El servidor carga el alfabeto desde un archivo, llamado “alfabeto.txt” y el hash desde otro llamado “hash.txt” y queda a la espera de las conexiones de los clientes en el puerto 4444. Tiene una lista de los clientes conectados y un tamaño máximo de clientes conectados (asignados por el sistema, con la variable de entorno del sistema UNIX FD_SETSIZE, que por defecto es 1024). Utilizaremos el operador select para esperar a las conexiones de los clientes y recibir sus operaciones.
Cuando recibe un paquete de un cliente verifica si posee la marca de palabra encontrada, sino remite a ese mismo cliente las próximas 10.000 palabras a procesar (cantidad por defecto, cambiable desde el código).

3.2 Cliente

El cliente se conecta al puerto 4444 del servidor y solicita al servidor remita su primer

paquete. En base a este, comienza a realizar las combinaciones posibles para saber cual es la palabra correspondiente a una determinada posición dentro de la combinatoria. Este trabajo lo realiza la función devuelvePalabra, que remite la posición y el alfabeto y retorna la palabra. Esto se debe a que es más sencillo remitir un número y no la palabra desde la cual comenzar.
El cálculo de la posición se realiza detectando cual es el largo de la palabra y luego dentro del largo, cual es la combinatoria correspondiente a esa posición.
Como ejemplo podríamos dar un alfabeto de tres letras, abc.
La primera posición sería la letra a, la segunda la letra b y la tercera la letra c. Para la cuarta posición la combinatoria daría aa, para la quinta ab, y así sucesivamente.
A esta palabra se le realiza el hash mediante la función crypt, y compara con el texto enviado dentro del hash por el servidor. De ser igual pone una marca en el paquete y remite al servidor para que este sepa que terminó.

3.3 Sobre la función Crypt
Esta función se puede consultar escribiendo man crypt en la consola bajo cualquier sistema operativo UNIX. Toma dos argumentos, el texto y la semilla.

char *crypt(const char *key, const char *salt);

key es el texto a encriptar, salt es la semilla que va a ser usada para producir las diferentes versiones encriptadas del mismo texto. La función retorna un puntero char al hash de key encriptado con la semilla salt. Si cambiamos el valor de salt de la función nos cambiará el hash de la clave, por lo tanto es conveniente no cambiar el salt. Digamos que es la clave publica de nuestro protocolo de hasheo. Evidentemente existen muchísimas combinaciones de salt, nosotros utilizaremos solamente una que nos indicará que encriptamos en MD5. Evitaremos pensar que en un sistema UNIX salt es desconocido, porque ahí deberíamos probar (además del las millones de combinaciones de palabras) las millones de combinaciones de semillas.

De la definición de crypt en su pagina man se puede extraer:

ID	Method
1	MD5
2a	Blowfish (not in mainline glibc; added in some Linux distributions)
5	SHA-256 (since glibc 2.7)
6	SHA-512 (since glibc 2.7)

por lo tanto una semilla del tipo '1' encripta en modo md5. También se puede encriptar de la forma '1UnaCadenaEnEspecial' lo cual generaría otro hash en md5 de la misma entrada para key.
Como ejemplo:
crypt(“hola”,”aa”) → “aaPwj9XL9opYuE”
crypt(“hola”,”bb”) → “bbRmaslkEk86ao”

Crypt no está recomendado como función de hash hoy en día por la mayoría de las aplicaciones, nosotros tomamos esta función como el ejemplo más básico de hashing. Muchas implementaciones de crypt no tienen vigencia porque no permiten que una clave tenga más de 8 caracteres (los demás los descarta). Para más información consultar en la web, existen infinidad de informes y librerías sobre seguridad de datos.

3.4 Protocolo
Para nuestro proyecto servirá un protocolo corto y de entendimiento fácil. Consta de las siguientes partes:

Número de paquete: el servidor impondrá el numero de paquete para (más

adelante) llevar a cabo un seguimiento de los paquetes a cada cliente, así también para distinguir unos de otros. Es el identificador único de cada paquete.

Alfabeto: Será la cadena desde la cual se generen las palabras, no se realiza ningún control sobre si hay letras repetidas.

Hash: Texto del cual se quiere encontrar la palabra origen.

Inicio: Número entre 1 y 2^{64}, denota la posición desde la cual se debe empezar a recorrer el vector formador de palabras.

Fin: Número entre 1 y 2^{64}, denota la posición hasta la cual se debe recorrer el vector formador de palabras.

Resultado: Forma de comunicación del cliente al servidor. Lleva el número que dio el resultado positivo.

4. Lógica de la Aplicación:

Una vez que se inicia el servidor, el cliente se conecta y le envía un paquete indicándole que está dispuesto a leer y procesar, esto es, en el protocolo van todos los campos vacíos. El servidor una vez que recibe este paquete genera las operaciones que le corresponde al cliente y se las envía.

El cliente recibe estos segmentos (inicio y fin) y comienza a iterar en las miles de combinaciones que existen dentro de ese inicio y fin. En cada iteración hace el hash de ésta y compara el hash originalmente pasado en el protocolo con el que resulta de la combinación. En caso de coincidir, detiene el procesamiento de datos, pone en resultado la combinación acertada y remite el paquete al servidor. En caso de no haber encontrado el hash (se da por sobreentendido que no hay más combinaciones) se devuelve el mismo paquete que recibió, para así ponerse a esperar un nuevo paquete del servidor con más segmentos inicio y fin.