Proyecto de Desarrollo e Investigación

Quebrado de hash MD5 bajo el paradigma cliente-servidor

Nicolás Alfonsín (nicolasalfonsin@gmail.com)
Diego Meretta (santicluke@gmail.com)
Ramiro Pugh (rampugh@gmail.com)

Trabajo de Investigación desarrollado en la cátedra Internetworking, Universidad Técnica Nacional, Facultad Regional la Plata, La Plata, Buenos Aires, Argentina

Resumen. Información teórica y desarrollo de un protocolo, cliente y servidor. El objetivo es señalar al usuario que la distribución de carga y actividades influye drásticamente en el rendimiento general del conjunto.

Palabras Claves: Cliente, Servidor, Hash, MD5, Crack

1. Introducción

El **hash** es un número (normalmente natural) que identifica a un documento, archivo o registro.

La **función de hash** es un método que se utiliza para realizar la comprobación de un documento, archivo o registro de manera casi univoca a través de lo que se llama suma de verificación. Las funciones de hash son excelentes para verificación ya que en su mayoría son de una vía, esto quiere decir que una vez encriptado, es imposible a partir de un hash saber el texto plano con la misma función.

Una de las funciones de hash más conocidos y utilizados es **MD5**, abreviatura de *Message-Digest Algorithm 5.*

Este algoritmo era el utilizado para los sistemas UNIX (y aún lo hacen) para guardar las contraseñas de los usuarios. Cuando un usuario escribe la clave para el login, el sistema genera el hash con la contraseña que uno escribió y la compara con la clave que se posee almacenada. En ningún momento se guarda el texto plano (4) de la contraseña guardada.

En este sentido, un atacante que toma una computadora y roba el archivo con las contraseñas (/etc/passwd en el caso de sistemas operativos UNIX) no sabe inmediatamente cuál es el texto legible de las contraseñas guardadas por los usuarios. Acá intervienen las múltiples herramientas que existen hoy en día para averiguar los textos a partir de un determinado hash. Se pueden encontrar en internet bases de datos con millones de textos y sus MD5 correspondiente.

Nos centraremos en este proyecto en la funcion 'crypt' (1) la cual es la usada por los primeros sistemas UNIX y por nuestro proyecto. Cuando esta función fue diseñada en el año 1976, las computadores de ese entonces solo podían aplicar la función de hash cuatro claves por segundo. Hoy en día una PC de escritorio puede llegar a realizar tres

millones de funciones crypt por segundo. Aquí hacemos incapié en dejar de lado las consideraciones de usar procesadores GPU los cuales superan aún más las prestaciones de un procesador normal.

(1) Para más información escribir en consola UNIX: man crypt.

De todas formas, tres millones de combinaciones no siguen siendo excesivamente rápidas. Usando mayúsculas, minúsculas y caracteres numéricos existen más de 62^8 combinaciones posibles de ocho caracteres. Esto significa que se tardarían unos 900 días de en quebrar por fuerza bruta (utilizar todas las combinaciones) utilizando una PC de escritorio.

2. Motivación

Es por eso que realizamos este proyecto; para comprobar y tomar datos de que distribuyendo las operaciones en múltiples clientes se puede lograr una drástica reducción de operaciones, es decir, que no sea un solo sistema el encargado de considerar todas las operaciones sino un sistema distribuido de computadoras que se encarguen conjuntamente del gran procesamiento que se necesita.

El gran avance que se ve hoy en día en materia de seguridad siempre es ofuscado por los grandes robos de información que se producen constantemente por la ignorancia de los usuarios, o la gran credibilidad que poseen hacia enlaces que les llegan por correos electrónicos o en los mismos enlaces de sus navegadores.

¿Pero qué pasa si los grandes comprometidos no son los usuarios finales sino que son los administradores de los sistemas, los dueños de toda la información que está en sus bases de datos (teóricamente) protegidas?

Nuestra investigación apunta a estos datos que son robados (casi constantemente) de las bases de datos de los servidores. Estos datos de por sí no tienen valor alguno más que servir para enviar correos basura, al tener el correo electrónico y el nombre de usuario. Pero si la base de datos robada contenía contraseñas, ahí el problema es otro y ése es nuestro punto de estudio.

Quizás el usuario común no sepa que las claves dentro de las bases de datos se encuentran con una función de hash, pero con mucha paciencia y rápidas operaciones de cómputo se puede obtener el texto plano de la clave de ese usuario. Nuestra idea es verificar que una contraseña puede ser violada indefectiblemente si se posee los conocimientos y gran capacidad de cómputo.

3. Estado del Arte

En la actualidad se pueden encontrar páginas web, como http://md5crack.com/crackmd5.php y http://bokehman.com/cracker/; software libre optimizado para GPUs, como http://bvernoux.free.fr/md5/index.php; software libre optimizado para GPUs y para varios cores, como http://project-rainbowcrack.com/; y software privativo que realiza la misma tarea con un grado mucho mayor de optimización; como ser http://www.elcomsoft.com/lhc.html.

Pero ninguno de estos programas tiene la posibilidad de combinar las técnicas de optimización para una sola CPU con el poder de la distribución de computo a través de granjas de procesamiento o de una red empresarial.

Como herramienta para la distribución de computo en varios CPUs podemos mencionar a BOINC, http://boinc.berkeley.edu/, que es una herramienta que permite realizar todos los tipos de optimizaciones que hemos mencionado (varios cores, GPUs y otros) y permite realizar la distribución que estamos estableciendo en este documento. En este caso se declino de utilizar esta herramienta dado que el fin del proyecto era trabajar con los protocolos de red directamente y no a través de una abstracción como lo hace

BOINC, herramienta que a su vez necesita un amplio estudio dado su gran potencial.

4. Arquitectura del Proyecto

Nuestro software distribuido usará un diseño relativamente estándar cliente-servidor y está íntegramente programado en el lenguaje C.

La comunicación utilizará sockets TCP para la comunicación con los nodos.

El servidor maestro central, será el responsable de la coordinación de los paquetes, la segmentación de los intervalos, el procesamiento de envíos y lectura de las respuestas.

El cliente será el encargado de tomar los segmentos y generar todas las combinaciones posibles hasta llegar al fin del intervalo recibido. En caso de encontrar la contraseña se le avisa al servidor.

4.1 Servidor

El servidor tendrá cuatro funcionalidades importantes: administración de clientes, generar los segmentos, enviar paquetes y recibir respuestas.

- Administración de clientes: el servidor se encargará de administrar los clientes conectados y llevar una lista de éstos.
- Envío y Recepción de Paquetes: los clientes al conectarse y al terminar de procesar los segmentos le comunicarán al servidor su estado. El servidor tomará medidas según las respuestas del cliente. En el caso de que el cliente recién se haya conectado, le envía el paquete a manejar. Si el cliente ha dado una respuesta sobre un procesamiento el servidor verifica la información que le ha comunicado el cliente y comprueba que el cliente no haya encontrado la clave. De ser así, envía un nuevo segmento para que el cliente opere y retorne las operaciones.
- Generación de Segmentos: se segmentarán las operaciones en cantidad de combinaciones. Cada segmento poseerá 10.000 combinaciones posibles. Como otro objeto de estudio se podría comprobar un punto de equilibrio entre la cantidad operaciones y numero de clientes.

4.2 Cliente

El cliente tiene la única funcionalidad de recibir los segmentos, generar todos las combinaciones en el segmento y comprobar el hash que devuelve la combinación con el hash pasado desde el servidor. De haber coincidido, se le devuelve al servidor un aviso de encuentro y se le pasa en el protocolo la
combinación encontrada. En caso contrario se devuelve al servidor el mismo paquete que recibió.

4.3 Protocolo

El protocolo cuenta con 5 campos. Estos campos son:

• Alfabeto: aquí se pasa al cliente el conjunto de caracteres y el orden en que se debe buscar dentro de éste.

• Inicio: el inicio del segmento.

• Fin: el fin del segmento.

• Hash: es el hash a comprobar, de ser igual el hash de una combinación con éste, se termina la operación y se da por finalizada la búsqueda.

• Resultado: forma de comunicación de opciones y errores. En caso de encontrarse el resultado, lleva la posición de la cadena favorable.

5. Evaluación del Sistema

En este punto realizamos un análisis del sistema para identificar su rendimiento y comprender su alcance.

Entorno: las pruebas se realizaron en procesadores de 64 bits corriendo a 2,9 Ghz.

Datos utilizados:
Alfabeto: abcdefghijklmnopqrstuvwxyz
Hash: $1$$mIq1jVE.bnCS7tDPq23a40 (pepe)
Cantidad de claves a procesar: 285.017

Dado que el sistema pretende llevar la carga a los clientes y no dejarla en el servidor, evaluamos con 3 clientes la formación de los paquete que se entrega, modificando la cantidad (modificable desde le código, línea 144 del servidor) para poder comprender cual es un valor razonable de claves a entregar a cada cliente.

CCP: Cantidad de claves por paquete
TTR: Tiempo total de resolución
UCC: Utilización de CPU en los clientes

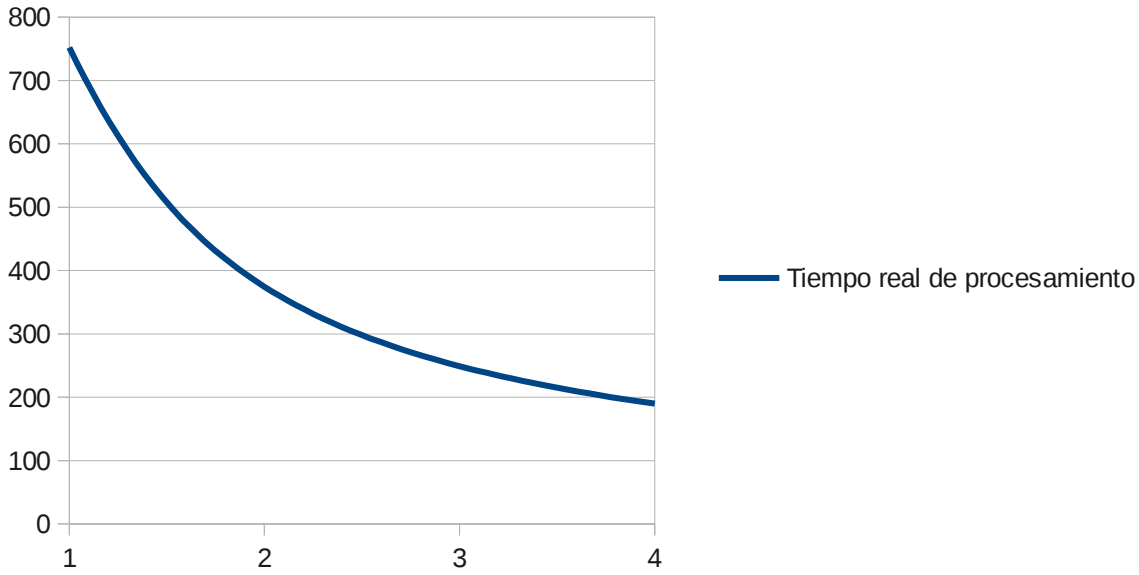
CCP	TTR	UCC
1	0m29.165s	~80%
5	0m19.068s	~95%
10	0m18.332s	100%

El análisis de los datos recogidos da cuenta de que aún utilizando una sola clave por paquete, el esfuerzo denotado por los clientes para la utilización de la función crypt, hace muy positivo la utilización de una infraestructura de red para averiguar hashes de manera distribuida.
Dado que este valor es realmente bajo, y con la espera de que podamos probar el servidor en un entorno realmente grande (más de 1.000 maquinas), el valor elegido para enviar a cada cliente será de 10.000 claves por paquetes, esto hace que el servidor quede más ocioso y pueda atender mejor a cada nuevo cliente.

Con este nuevo valor vamos ahora a realizar una prueba de estrés, dando al servidor un número variable de clientes, para poder analizar la escalabilidad de software.

Alfabeto:
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ01234567890
Hash: $1$$mIq1jVE.bnCS7tDPq23a40 (pepe)
Cantidad de claves a procesar: 3.833.465

Cantidad de Clientes	Tiempo real de procesamiento
1	12m31.963s
2	6m14.575s
3	4m9.148s
4	3m9.860s



Como se puede observar en el gráfico, la cantidad de clientes hace que se reduzca linealmente el tiempo demorado en finalizar la tarea, que era la razón principal de nuestro proyecto.

Conclusión:
Viendo las herramientas que existen en Internet, encontramos en ese aspecto un vacío para este tipo de herramientas, o vimos que las que existen son pagas y no accesibles a cualquiera.
Dado que este es un proyecto estudiantil, pudimos demostrar alta escalabilidad a la cantidad de clientes y un muy buen rendimiento para la comparación de claves MD5.
Como camino a recorrer queda por delante la optimización para GPUs (técnica reciente) pero no excluyente de la actual implementación, así como también la utilización de distintos algoritmos, como podría ser SHA-1, archivos zip, etc. Es decir, todo tipo de contraseña posible.