Encontrando colisiones

Este documento no pretende ser un curso exhaustivo, en el mejor de los casos, únicamente puede considerarse como un conjunto de notas complementarias en alguna asignatura. Por supuesto, todo documento es susceptible de mejora, cualquier sugerencia o comunicación de error u omisión será bienvenida.

El uso de funciones resumen en protocolos criptográficos está basado en la complejidad de encontrar colisiones para una entrada o resumen dados. La técnica que se expone aquí se basa en compensar el tiempo de computación necesario para obtener una colisión mediante un preproceso, creando un balance entre memoria utilizada y tiempo de computación, permitiendo la búsqueda eficiente de colisiones.

Rainbow attack

Sistema originalmente diseñado para la búsqueda de colisiones de passwords. Dado un password de entrada, el método reduce la complejidad de la búsqueda almacenando información suficiente para recuperar un password (no necesariamente el recibido como entrada) con el mismo resumen. Esta aproximación reduce la necesidad de almacenamiento de información (password + resumen) de los ataques basados en diccionario.

El preprocesamiento considera:

- Una función hash de n bits $h:\ D \rightarrow \{0,1\}^n$,
- Una función de reducción $r:\ \{0,1\}^n \rightarrow D$,

para construir un conjunto de n cadenas de passwords que comienzan con uno escogido al azar, que se resume (aplicando la función h) para después reducirlo, recodificándolo (aplicando la función r) para obtener de nuevo un posible password. La recodificaciónq tiene como objeto adaptar el resumen a las restricciones que habitualmente se aplican a la construcción de passwords. El proceso se repite hasta alcanzar una longitud t fijada a priori. De toda esta secuencia, únicamente se almancena sel primer y último passwords obtenidos. La Figura 1 esquematiza el proceso y el Algoritmo 1 lo describe algorítmicamente.

Habitualmente los passwords son resumidos, resumen que es almacenado de forma abierta y accesible en los sistemas. Dado el resumen de un password cualquiera p_0, puede utilizarse el vector rainbow para obtener otro password p' tal que $h(p_0) = h(p')$, lo que garantiza el acceso al sistema.

El proceso de búsqueda parte de $p = p_0$ y consiste en, buscar en la tabla una entrada $\langle b_i, e_i \rangle$ tal que e_i sea igual a $q = r(p)$. Si no se encuentra tal entrada, se repite el proceso

Algoritmo 1 Construcción de un vector rainbow

Entrada: Una función resumen h
Entrada: Una función recodificante r
Entrada: t longitud de la secuencia
Entrada: n número de entradas
Salida: Un vector rainbow para la función h.

Método
Mientras La tabla no contenga n entradas **hacer**
Escoger P_1 un password al azar.
for $i = 1$ **to** $t - 1$ **hacer**
$P_{i+1} = r(h(P_i))$
FinPara
Almacenar $\langle P_1,\ P_t \rangle$ en la tabla
FinMientras
FinMétodo.

Algoritmo 2 Búsqueda de colisiones utilizando un vector rainbow

Entrada: Una password p_0
Entrada: Una función resumen h
Entrada: Una función recodificante r
Entrada: t longitud de la secuencia
Entrada: Un vector rainbow para la función h.
Salida: p tal que $h(p_0) = h(p)$ o $ERROR$

Método
$p = p_0$, $q = r(p)$
for $i = 1$ **to** t **hacer**
Si Existe en la tabla $\langle b, e \rangle$ tal que $e = q$ **entonces** BreakFor
$p = h(q)$
FinPara
Si $i == t$ **entonces**
Devolver $ERROR$
FinSi
$p = b$
Mientras $h(p) \neq p_0$ **hacer**
$p = r(h(p))$
FinMientras
Devolver p
FinMétodo.

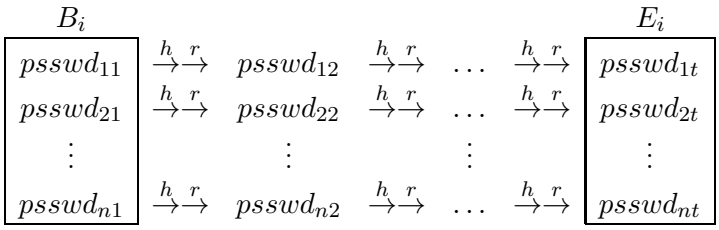

Figura 1: Esquema de construcción de un vector rainbow. Los cuadros indican la información que se almacena en la tabla.

considerando el resumen de q, esto es haciendo $p = h(q)$. El proceso continua mientras no se encuentre la entrada en la tabla o se alcance la longitud t.

Si el proceso ha detectado que la entrada $\langle b_i, e_i \rangle$ es tal que e_i colisiona con alguna de las recodificaciones $r(p)$, entonces es posible obtener un password válido partiendo de b_i, aplicando sucesivamente el resumen y la recodificación. El Algoritmo 2 describe el proceso.

Consideraciones de implementación

Consideremos un dominio de D donde la logitud máxima de un password sea l, de este modo existen $|D|^l$ passwords distintos. Ya que cada entrada de la tabla permite recuperar, en principio, t passwords distintos, abordar un ataque basado en un vector rainbow necesitaría almacenar un número de entradas:

$$n = \frac{|D|^l}{t},$$

El inconveniente fundamental se debe a la probabilidad de encontrar colisiones mientras se está construyendo el vector rainbow, lo que reduce el número de passwords efectivos que pueden reconstruirse a partir del vector rainbow. Se han propuesto distintas formas de tratar este inconveniente: considerando distintas funciones de recodificación, obteniendo las que se conocen como tablas rainbow, estableciendo *puntos de sincronización* donde la secuencia finaliza independientemente de haber alcanzado la longitud máxima, o bien construyendo en paralelo distintas estructuras rainbow.

Es importante tener en cuenta que la función resumen es un parámetro de entrada en este tipo de ataque, por lo que deberían construirse distintos vectores/tablas rainbow para cada función resumen objeto de interés.

Técnicas como el *salting* hacen necesario multiplicar el número de tablas a construir, haciendo este tipo de ataque inviable.