Laboratorio de ataque de colisión MD5

1. Introducción

Una función hash unidireccional segura debe satisfacer dos propiedades: la propiedad unidireccional y la propiedad de resistencia a la colisión. La propiedad unidireccional asegura que, dado un valor hash h, no es computacionalmente factible encontrar una entrada M, tal que hash(M) = h. La propiedad de resistencia a la colisión asegura que no sea computacionalmente factible encontrar dos entradas diferentes M1 y M2, tales que hash(M1) = hash(M2).

Varias funciones hash unidireccionales ampliamente utilizadas tienen problemas para mantener la propiedad de resistencia a colisiones. En la sesión final de CRYPTO 2004, Xiaoyun Wang y sus coautores demostraron un ataque de colisión contra MD5 [1]. En febrero de 2017, CWI Amsterdam y Google Research anunciaron el ataque SHAttered, que rompe la propiedad de resistencia a colisiones de SHA-1 [3]. Si bien muchos estudiantes no tienen problemas para comprender la importancia de la propiedad unidireccional, no pueden comprender fácilmente por qué es necesaria la propiedad de resistencia a colisiones y qué impacto pueden causar estos ataques.

El objetivo de aprendizaje de este laboratorio es que los estudiantes comprendan realmente el impacto de los ataques de colisión y vean de primera mano qué daños pueden causarse si se rompe la propiedad de resistencia a la colisión de una función hash unidireccional ampliamente utilizada. Para lograr este objetivo, los estudiantes deben lanzar ataques de colisión reales contra la función hash MD5. Usando los ataques, los estudiantes deberían poder crear dos programas diferentes que comparten el mismo hash MD5 pero tienen comportamientos completamente diferentes. Este laboratorio cubre una serie de temas que se describen a continuación:

- Función hash unidireccional, MD5
- La propiedad de resistencia a colisiones
- Ataques de colisión

Lecturas. La cobertura detallada de la función hash unidireccional se puede encontrar en lo siguiente:

- Capítulo 22 del libro de SEED, Computer & Internet Security: A Hands-on Approach, 2nd Edition, por Wenliang Du. Consulte los detalles en https://www.handsonsecurity.net.

Ambiente de laboratorio. Este laboratorio se probó en nuestra máquina virtual Ubuntu 20.04 preconstruida, que se puede descargar desde el sitio web de SEED. El laboratorio utiliza una herramienta llamada “Fast MD5 Collision Generation”, que fue escrita por Marc Stevens. El nombre del binario se llama md5collgen en nuestra VM y está instalado dentro de la carpeta /usr/bin. Si no está allí, puede descargarlo del sitio web del laboratorio (dentro de Labsetup.zip).
Si está interesado en instalar la herramienta en su propia máquina, puede descargar el código fuente directamente desde https://www.win.tue.nl/hashclash/.

Agradecimientos Esta práctica de laboratorio se desarrolló con la ayuda de Vishtasp Jokhi, estudiante de posgrado en el Departamento de Ingeniería Eléctrica y Ciencias de la Computación de la Universidad de Syracuse.

2 tareas de laboratorio

2.1 Tarea 1: Generación de dos archivos diferentes con el mismo hash MD5

En esta tarea, generaremos dos archivos diferentes con los mismos valores hash MD5. Las partes iniciales de estos dos archivos deben ser iguales, es decir, comparten el mismo prefijo. Podemos lograr esto utilizando el programa md5collgen, que nos permite proporcionar un archivo de prefijo con cualquier contenido arbitrario. La forma en que funciona el programa se ilustra en la Figura 1. El siguiente comando genera dos archivos de salida, out1.bin y out2.bin, para un archivo de prefijo prefijo.txt dado:

```
$ md5collgen -p prefijo.txt -o out1.bin out2.bin
```

Figura 1: Generación de colisiones MD5 a partir de un prefijo

Podemos verificar si los archivos de salida son distintos o no usando el comando diff. También podemos usar el Comando md5sum para verificar el hash MD5 de cada archivo de salida. Consulte los siguientes comandos.

```
$ diff out1.bin out2.bin $ md5sum
out1.bin $ md5sum out2.bin
```

Dado que out1.bin y out2.bin son binarios, no podemos verlos usando un programa de visualización de texto, como cat o más; necesitamos usar un editor binario para verlos (y editarlos). Ya hemos instalado un software de edición hexadecimal llamado bless en nuestra máquina virtual. Utilice dicho editor para ver estos dos archivos de salida y describa sus observaciones. Además, debe responder a las siguientes preguntas:

– **Pregunta 1.** Si la longitud de su archivo de prefijos no es múltiplo de 64, ¿qué va a pasar?

– **Pregunta 2.** Cree un archivo de prefijo con exactamente 64 bytes, ejecute la herramienta de colisión nuevamente y vea qué sucede

– **Pregunta 3.** ¿Los datos (128 bytes) generados por md5collgen son completamente diferentes para los dos archivos de salida? Por favor identifique todos los bytes que son diferentes.

2.2 Tarea 2: comprensión de la propiedad de MD5

En esta tarea, intentaremos comprender algunas de las propiedades del algoritmo MD5. Estas propiedades son importantes para que podamos realizar más tareas en este laboratorio. MD5 es un algoritmo bastante complicado, pero de muy alto nivel, no es tan complicado. Como muestra la Figura 2, MD5 divide los datos de entrada en bloques de 64 bytes y luego calcula el hash iterativamente en estos bloques. El núcleo del algoritmo MD5 es una función de compresión, que toma dos entradas, un bloque de datos de 64 bytes y el resultado de la iteración anterior. La función de compresión produce un IHV de 128 bits, que significa "Valor hash intermedio"; esta salida luego se alimenta a la siguiente iteración. Si la iteración actual es la última, el IHV será el valor hash final.

La entrada IHV para la primera iteración (IHV0) es un valor fijo.

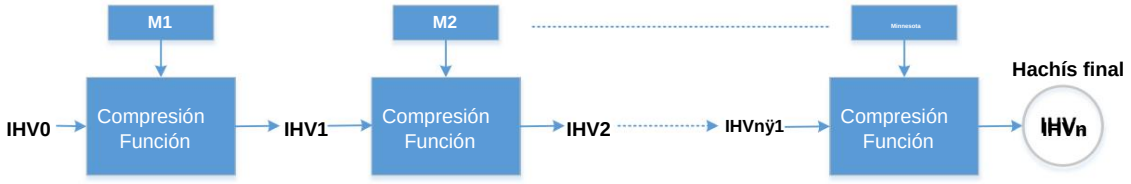

Figura 2: Cómo funciona el algoritmo MD5

Basándonos en cómo funciona MD5, podemos derivar la siguiente propiedad del algoritmo MD5: Dadas dos entradas M y N, si MD5(M) = MD5(N), es decir, los hashes MD5 de M y N son los mismos, entonces para cualquier entrada T, MD5(M k T) = MD5(N k T), donde k representa la concatenación.

Es decir, si las entradas M y N tienen el mismo hash, agregarles el mismo sufijo T dará como resultado dos salidas con el mismo valor hash. Esta propiedad es válida no solo para el algoritmo hash MD5, sino también para muchos otros algoritmos hash. Su trabajo en esta tarea es diseñar un experimento para demostrar que esta propiedad se cumple para MD5.

Puede usar el comando cat para concatenar dos archivos (archivos binarios o de texto) en uno. El seguimiento El comando concatena el contenido del archivo2 con el contenido del archivo1 y coloca el resultado en el archivo3.

```
$ gato archivo1 archivo2 > archivo3
```

2.3 Tarea 3: Generación de dos archivos ejecutables con el mismo hash MD5

En esta tarea, se le proporciona el siguiente programa en C. Su trabajo es crear dos versiones diferentes de este programa, de modo que el contenido de sus matrices xyz sea diferente, pero los valores hash de los ejecutables sean los mismos.

```
#incluir <stdio.h>

carácter sin firmar xyz[200] = {
  /* El contenido real de esta matriz depende de usted */ };


int principal() {

    ent yo;
    para (i=0; i<200; i++){
        printf("%x", xyz[i]);

    } printf("\n");
}
```

Puede optar por trabajar en el nivel del código fuente, es decir, generar dos versiones del programa C anterior, de modo que después de la compilación, sus archivos ejecutables correspondientes tengan el mismo valor hash MD5. Sin embargo, puede ser más fácil trabajar directamente en el nivel binario. Puede poner algunos valores arbitrarios en la matriz xyz, compilar el código anterior en binario. Luego puede usar una herramienta de edición hexadecimal para modificar el contenido de la matriz xyz directamente en el archivo binario.

Encontrar dónde se almacenan los contenidos de la matriz en el binario no es fácil. Sin embargo, si llenamos la matriz con algunos valores fijos, podemos encontrarlos fácilmente en el binario. Por ejemplo, el siguiente código llena la matriz con 0x41, que es el valor ASCII de la letra A. No será difícil ubicar 200 A en el binario.

```
carácter sin firmar xyz[200] = {
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
    0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, ...
    (omitido) ... 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,


}
```

Pautas. Dentro de la matriz, podemos encontrar dos ubicaciones, desde donde podemos dividir el archivo ejecutable en tres partes: un prefijo, una región de 128 bytes y un sufijo. La longitud del prefijo debe ser un múltiplo de 64 bytes. Consulte la Figura 3 para ver una ilustración de cómo se divide el archivo.

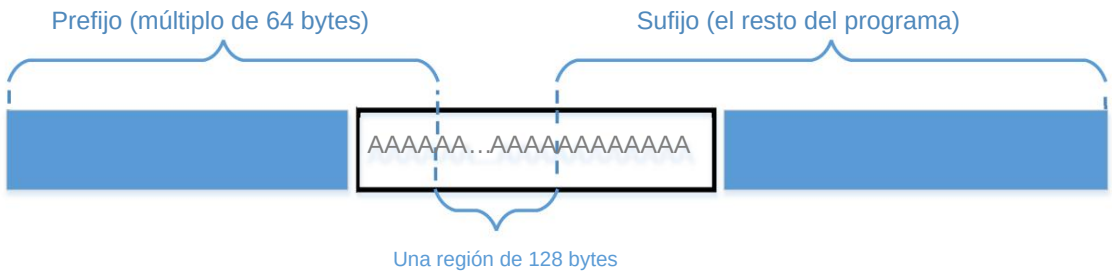

Figura 3: Divida el archivo ejecutable en tres partes.

Podemos ejecutar md5collgen en el prefijo para generar dos salidas que tengan el mismo valor hash MD5. Usemos P y Q para representar la segunda parte (cada una con 128 bytes) de estas salidas (es decir, la parte después del prefijo). Por lo tanto, tenemos lo siguiente:

```
MD5 (prefijo k P) = MD5 (prefijo k Q)
```

Con base en la propiedad de MD5, sabemos que si agregamos el mismo sufijo a las dos salidas anteriores, el los datos resultantes también tendrán el mismo valor hash. Básicamente, lo siguiente es cierto para cualquier sufijo:

```
MD5 (prefijo k P k sufijo) = MD5 (prefijo k Q k sufijo)
```

Por lo tanto, solo necesitamos usar P y Q para reemplazar 128 bytes de la matriz (entre los dos puntos de división), y podremos crear dos programas binarios que tengan el mismo valor hash. Sus resultados son diferentes, porque cada uno imprime sus propias matrices, que tienen contenidos diferentes.

Instrumentos. Puede usar bless para ver el archivo ejecutable binario y encontrar la ubicación de la matriz. Para dividir un archivo binario, existen algunas herramientas que podemos usar para dividir un archivo desde una ubicación particular. Los comandos de cabeza y cola son herramientas muy útiles. Puedes consultar sus manuales para aprender a usarlos.

Damos tres ejemplos a continuación:

```
$ cabeza -c 3200 a.out > prefijo $ tail -c 100
a.out > sufijo $ tail -c +3300 a.out > sufijo
```

El primer comando anterior guarda los primeros 3200 bytes de a.out en el prefijo. El segundo comando guarda los últimos 100 bytes de a.out en sufijo. El tercer comando guarda los datos desde el byte 3300 hasta el

final del archivo a.out al sufijo. Con estos dos comandos, podemos dividir un archivo binario en partes desde cualquier ubicación. Si necesitamos pegar algunas piezas, podemos usar el comando gato.

Si usa bless para copiar y pegar un bloque de datos de un archivo binario a otro archivo, el elemento de menú "Editar -> Seleccionar rango" es bastante útil, porque puede seleccionar un bloque de datos usando un punto de inicio y un intervalo, en lugar de contar manualmente cuántos bytes se seleccionan.

2.4 Tarea 4: hacer que los dos programas se comporten de manera diferente

En la tarea anterior, hemos creado con éxito dos programas que tienen el mismo hash MD5, pero sus comportamientos son diferentes. Sin embargo, sus diferencias están solo en los datos que imprimen; todavía ejecutan la misma secuencia de instrucciones. En esta tarea, nos gustaría lograr algo más significativo y significativo.

Suponga que ha creado un software que hace cosas buenas. Envía el software a una autoridad de confianza para obtener la certificación. La autoridad realiza una prueba exhaustiva de su software y concluye que su software realmente está haciendo cosas buenas. La autoridad le entregará un certificado, indicando que su programa es bueno. Para evitar que cambie su programa después de obtener el certificado, el valor hash MD5 de su programa también se incluye en el certificado; el certificado está firmado por la autoridad, por lo que no puede cambiar nada en el certificado o su programa sin invalidar la firma.

Le gustaría que la autoridad certifique su software malicioso, pero no tiene ninguna posibilidad de lograr ese objetivo si simplemente envía su software malicioso a la autoridad. Sin embargo, ha notado que la autoridad usa MD5 para generar el valor hash. Tienes una idea. Planeas preparar dos programas diferentes. Un programa siempre ejecutará instrucciones benignas y hará cosas buenas, mientras que el otro programa ejecutará instrucciones maliciosas y causará daños. Encuentra una manera de hacer que estos dos programas compartan el mismo valor hash MD5.

Luego envía la versión benigna a la autoridad para la certificación. Dado que esta versión hace cosas buenas, pasará la certificación y obtendrá un certificado que contiene el valor hash de su programa benigno.
Debido a que su programa malicioso tiene el mismo valor hash, este certificado también es válido para su programa malicioso. Por lo tanto, ha obtenido con éxito un certificado válido para su programa malicioso. Si otras personas confiaron en el certificado emitido por la autoridad, descargarán su programa malicioso.

El objetivo de esta tarea es lanzar el ataque descrito anteriormente. Es decir, debe crear dos programas que compartan el mismo hash MD5. Sin embargo, un programa siempre ejecutará instrucciones benignas, mientras que el otro programa ejecutará instrucciones maliciosas. En su trabajo, no es importante qué instrucciones benignas/maliciosas se ejecutan; es suficiente demostrar que las instrucciones ejecutadas por estos dos programas son diferentes.

Pautas. Crear dos programas completamente diferentes que produzcan el mismo valor hash MD5 es bastante difícil. Los dos programas de colisión de hash producidos por md5collgen deben compartir el mismo prefijo; además, como podemos ver en la tarea anterior, si necesitamos agregar algún sufijo significativo a las salidas producidas por md5collgen, el sufijo agregado a ambos programas también debe ser el mismo. Estas son las limitaciones del programa de generación de colisiones MD5 que utilizamos. Aunque existen otras herramientas más complicadas y avanzadas que pueden eliminar algunas de las limitaciones, como aceptar dos prefijos diferentes [2], exigen mucha más potencia informática, por lo que están fuera del alcance de esta práctica de laboratorio. Necesitamos encontrar una manera de generar dos programas diferentes dentro de las limitaciones.

Hay muchas maneras de lograr el objetivo anterior. Proporcionamos un enfoque como referencia, pero se anima a los estudiantes a que presenten sus propias ideas. Los instructores pueden considerar recompensar a los estudiantes por sus propias ideas. En nuestro enfoque, creamos dos matrices X e Y. Comparamos el contenido de estas dos matrices; si ellos

son iguales, se ejecuta el código benigno; de lo contrario, se ejecuta el código malicioso. Consulte el siguiente pseudocódigo:

```
matriz X;
matriz Y;

principal()
{
    if (los contenidos de X y los contenidos de Y son los mismos) ejecute código
            benigno;
    más
            ejecutar código malicioso;
    devolver;
}
```

Podemos inicializar las matrices X e Y con algunos valores que pueden ayudarnos a encontrar sus ubicaciones en el archivo binario ejecutable. Nuestro trabajo es cambiar el contenido de estas dos matrices, para que podamos generar dos versiones diferentes que tengan el mismo hash MD5. En una versión, los contenidos de X e Y son los mismos, por lo que se ejecuta el código benigno; en la otra versión, los contenidos de X e Y son diferentes, por lo que se ejecuta el código malicioso. Podemos lograr este objetivo utilizando una técnica similar a la utilizada en la Tarea 3. La Figura 4 ilustra cómo se ven las dos versiones del programa.

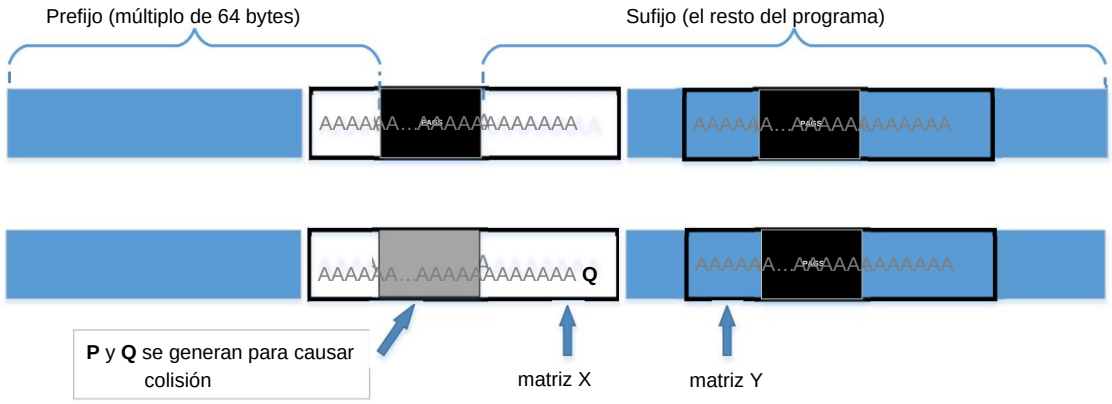

Figura 4: Un enfoque para generar dos programas de colisión de hash con diferentes comportamientos.

De la Figura 4, sabemos que estos dos archivos binarios tienen el mismo valor hash MD5, siempre que P y Q se generen en consecuencia. En la primera versión, hacemos que los contenidos de las matrices X e Y sean iguales, mientras que en la segunda versión, hacemos que sus contenidos sean diferentes. Por lo tanto, lo único que necesitamos cambiar es el contenido de estas dos matrices y no hay necesidad de cambiar la lógica de los programas.

3 Presentación

Debe enviar un informe de laboratorio detallado, con capturas de pantalla, para describir lo que ha hecho y lo que ha observado. También debe proporcionar una explicación a las observaciones que son interesantes o sorprendentes.
Enumere también los fragmentos de código importantes seguidos de una explicación. Simplemente adjuntar el código sin ninguna explicación no recibirá créditos.

Referencias

[1] John Black, Martin Cochran y Trevor Highland. Un estudio de los ataques md5: conocimientos y mejoras. En las Actas de la 13.ª Conferencia Internacional sobre Cifrado Rápido de Software, FSE'06, páginas 262–277, Berlín, Heidelberg, 2006. Springer-Verlag.

[2] Marc Stevens. En colisiones para md5. Tesis de maestría, Universidad Tecnológica de Eindhoven, 6 2007.

[3] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini y Yarik Markov. la primera colisión para SHA-1 completo. CWI Ámsterdam y Google Research, https://shattered.io/, 2017.