ESCENARIOS TÍPICOS DE FALLAS DE SEGURIDAD RELACIONADAS CON LA INTEGRIDAD DE DATOS

Resumen / *Abstract*
Se presentan tres escenarios típicos relacionados con violaciones a la integridad de datos. Estos escenarios cubren situaciones comunes que han olvidado un importante atributo de la seguridad. Se presentan además las funciones de *hash* como una alternativa para detectar los ataques a la integridad de los datos.

Three typical scenarios related with data integrity violations are described. They cover common situations that forgot an important security issue. Application of hash functions, as a way to detect data integrity attacks, is presented.

Palabras clave / *Key words*
Funciones de *hash*, criptografía, integridad de datos

Hash functions, cryptography, data integrity

INTRODUCCIÓN

El tema de la seguridad informática en la época actual, hace pensar de manera casi total, en virus, ataques a la autenticación o confidencialidad de las redes y sistemas informáticos, entre otras refinadas modalidades de cultivo del delito que se acrecienta sobre estos sistemas.

Existen, sin embargo, atributos clásicos de la seguridad que al parecer han sido frecuentemente obviados y hasta olvidados en el desarrollo de sistemas de hardware y software.

Uno de ellos, la integridad de datos puede dar al traste con muchas modalidades de ataques que son exitosos debido a que la mayoría de las implementaciones obvian la realización de chequeos de control integridad a paquetes de datos o archivos.

En el artículo que se presenta se distinguen tres escenarios reales, descritos de manera crítica a fin de comprender cuán importante es reconsiderar el atributo de la integridad.

En correspondencia con un estilo empleado en la descripción de fallas de seguridad, se hará énfasis en la problemática concreta sin identificar de manera directa estos escenarios cuya aparente generalidad es lamentable.

Se propone además una solución general basada en funciones de compresión* criptográficas que constituyen una alternativa fiable a la detección ante tales ataques.

***Roberto Sepúlveda Lima**,Ingeniero en Máquinas Computadoras, Doctor en Ciencias Técnicas, Profesor Titular. Centro de Estudios de Ingeniería de Sistemas (CEIS), Instituto Superior Politécnico José Antonio Echeverría, cujae, Ciudad de La Habana, Cuba*
e-mail:sepul@ceis.cujae.edu.cu
***Frank David Abá Medina**, Ingeniero en Automática, CEIS, Instituto Superior Politécnico José Antonio Echeverría, cujae, Ciudad de La Habana, Cuba*
e-mail:abacarr@yahoo.com
***Léster Crespo Sierra**,Ingeniero en Automática, CEIS, Instituto Superior Politécnico José Antonio Echeverría, cujae, Ciudad de La Habana, Cuba*
e-mail:ester@electrica.cujae.edu.cu

Recibido:Julio del 2003
Aprobado:Octubre del 2003

* Conocidas como funciones de *hash*, por su denominación en idioma inglés.

TRES ESCENARIOS DE FALLAS DE SEGURIDAD

A continuación se describen tres escenarios reales de implementaciones que obvian el aspecto de la integridad de los datos y que dejan abierto un espacio para ataques basados en alterar el contenido de segmentos de información, que no son detectados por ninguna componente de seguridad del sistema.

Se obviará realizar referencias concretas de las situaciones analizadas, análisis a fin de no hacer públicas evidentes fallas de seguridad, sin embargo, los autores esperan que muchos desarrolladores se sientan aludidos y se inclinen a tomar las medidas que correspondan.

Primer escenario: La integridad de una base de datos

Una institución universitaria trabaja durante varios años sobre un sistema de gestión que emplea una base de datos centralizada. Se confrontan frecuentes problemas culturales para el manejo del sistema que ponen en discusión a los usuarios frente a los desarrolladores.

La base de datos diseñada contiene información sobre los estudiantes, incluidos datos de sus resultados académicos y se ha implementado empleando como manejador de bases de datos el sistema Microsoft Access.

Los alumnos que forman parte de la investigación están preparados en el uso de la informática y estos modifican la información de la base de datos empleando herramientas externas al sistema, entre ellas el propio manejador de bases de datos.

El sistema es incapaz de evitar que la base sea modificada por el exterior y sin mucho esfuerzo puede ser alterada incluso, desde un editor de textos, como el **bloc de notas**.

Un nuevo equipo de desarrolladores, pretende reiniciar el trabajo y considera reutilizar el diseño de la base de datos y su contenido. Por ello solicitan la base de datos a los responsables del proyecto anterior.

Sobre el particular quedan las interrogantes que siguen:

- ¿Cómo puede establecerse el estado de la base de datos que se entrega, si es vulnerable a ataques de integridad?
- ¿Qué grado de responsabilidad tiene el equipo que hace entrega de la base de datos, si los receptores expresan una reclamación a posteriori porque los datos no se corresponden con la realidad que modelan?

Situaciones similares ocurren con productos comerciales que se encuentran en uso en muchas instituciones que manejan información en bases de datos.

Cuando se cuestiona a usuarios y desarrolladores si en el diseño del sistema se ha tenido en cuenta el aspecto de la integridad de los datos, se puede determinar rápidamente que el tema de la seguridad, en general, y el de la integridad en particular ha sido obviado.

En el mejor de los casos, se ha pretendido obtener un nivel mínimo de control de autenticación, por medio de una contraseña. Este aspecto también requiere de controles de integridad, pero este tema será tratado en el próximo escenario.

Segundo escenario: Contraseña de acceso almacenada en el código

A veces se considera que introducir una caja de diálogo en la aplicación y exigir que el usuario emplee una contraseña para autenticarse es seguridad suficiente para un sistema informático.

Lamentablemente, estos enfoques se han empleado, en general, sin la profundidad adecuada en los ámbitos académicos y han trascendido al ejercicio de la profesión en los desarrolladores.

En muchas ocasiones, incluso, se ha fundamentado la seguridad del acceso en correspondencia con la longitud y la complejidad al ataque a fuerza bruta, lo cual es parcialmente cierto.[1,2]

Se presentan dos problemas que pueden ser fundamentales.

Existe la posibilidad de hurgar en el código ejecutable hasta encontrar el chequeo de la contraseña y deshabilitar el chequeo, lo cual permite atacar aún sin el conocimiento de la contraseña, o simplemente, encontrar la contraseña almacenada en texto claro en el ejecutable.

En una implementación analizada, se observó la opción de conectarse como invitado con la posibilidad de establecer para tal fin un usuario y contraseña.

En ese caso fue posible emplear la herramienta de búsqueda del sistema operativo para encontrar un archivo que contuviera la contraseña del invitado. Esto permitió abrir el archivo de marras con un editor de textos y descubrir la contraseña de administrador del sistema.

En cualquier caso, se pueden emplear variantes más refinadas para realizar ataques a contraseñas, sin necesidad de enfrentar la presunta complejidad de un ataque a fuerza bruta.

Se pueden emplear muchas herramientas libres que permiten descifrar el código ejecutable, observar detalles textuales (generalmente conservados en claro por los compiladores), y deshabilitar el chequeo de contraseñas, a veces sin necesidad de conocerlas.[3]

Se considera que la mayor gravedad del caso, radica en que con suficiente cuidado puede modificarse el ejecutable, alterando su contenido y burlando su integridad.

En lo adelante, los usuarios, no detectarán que están ejecutando un programa que no es exactamente el de las corridas anteriores y el atacador seguirá accediendo como un usuario habitual.

Tercer escenario: Ataque de integridad relacionado con las llaves de hardware

Las llaves de hardware se basan en un interesante presupuesto, que es el de posibilitar la copia de software pero impedir la ejecución sin una componente de hardware externa conectada a la máquina, que es la llave usualmente insertada en el puerto paralelo o en el puerto USB.*

*USB: Universal Serial Bus, de su denominación en idioma inglés.

Las llaves de hardware actuales tienen un adecuado nivel de ntegración y permiten almacenar en ellas diferentes piezas para a seguridad del sistema de software al cual se asocian, ya sean contraseñas, contadores, etcétera.

Los autores han realizado un ataque que se puede considerar ngenuo, dado que se basa en respetar un enfoque planteado por el fabricante de la llave para alcanzar seguridad y realizar el ataque sustituyendo las bibliotecas de enlaces dinámicos (DLL)* que se brindan para el desarrollador.

Así las cosas, basta con simular la presencia de las DLL reales, con llamados a las funciones previstas que permitan devolver el resultado esperado aún en ausencia de la llave.

Este ataque permite acceder al sistema protegido sin necesidad de tocar la llave ni alterar el ejecutable, lo cual constituye un ataque a la integridad del conjunto programa ejecutable-bibliotecas del fabricante - llave de hardware.

Las indicaciones del fabricante,[4] solamente establecen la necesidad de asegurarse del empleo de las DLLs adecuadas a fin de garantizar la seguridad buscada.

Esta recomendación es insuficiente para el desarrollador medio, que no debe confiar solamente en la funcionalidad y que debe incorporar mecanismos de control de integridad de mayor complejidad.

Este ataque particular, no permite generalizar esta situación a todas las variantes de llaves de hardware empleadas (ni lo pretende), pero permite asegurar que un chequeo de integridad de archivos componentes de un sistema informático que se base únicamente en el nombre, localización o tamaño de los archivos empleados es insuficiente.

Quizás sea interesante profundizar en el tema del empleo de bibliotecas de enlaces dinámicos como solución a funcionalidades relacionadas con la seguridad.

FUNCIONES DE HASH: UNA ALTERNATIVA AL CONTROL DE INTEGRIDAD

Aunque pueda aparecer sorprendente muchas implementaciones estándares como algunos protocolos de comunicación emplean como mecanismo de control de integridad de los paquetes de datos a la suma de comprobación[5], el cual es probadamente débil.

Muy pocas implementaciones, sin embargo, recurren a los códigos de redundancia cíclica (CRC), que constituyen una alternativa bastante fiable como control de integridad.[6]

Existen, sin embargo, interesantes resultados asociados a las funciones de compresión unidireccional (*hash*), que se basan en el empleo de un algoritmo criptográfico simétrico.[2]

Una función de compresión unidireccional es aquella que permite dado un bloque de información *M* de cualquier longitud, comprimir este a un bloque *h* de longitud fija *k*, comúnmente de 64, 128 o 256 bit.[2,7]

La función f de compresión unidireccional puede escribirse como $h = f(M)$ y debe cumplir las siguientes propiedades:

1. Dado *M* es muy fácil calcular *h*.
2. Dado *h* es muy difícil obtener M tal que $h = f(M)$.
3. Dado *M* es muy difícil generar otro mensaje *M*' tal que $f(M') = f(M)$.

Las propiedades 1 y 2 brindan el sentido de la unidireccionalidad deseable en una función de *hash*.** La tercera propiedad exige que la función se libre de colisiones.[2]

En realidad, no es difícil encontrar una función de compresión que obtenga a la salida un bloque de longitud fija para una entrada de longitud arbitraria.

Para un paquete de información de determinada longitud Q, bastaría dividirlo en k subbloques M_i de longitud fija, combinar cada bloque M_i del mensaje original con el valor de hash h_i del bloque anterior para obtener, su propio valor de compresión. Esto es:

$$h_i = f(M_i, h_i\text{-}1) \quad \text{para } i=1,2,3,....k\text{-}1. \qquad ...(1)$$

donde:

$h_1 = f(M_1, h_0)$, tal que h_0 es una cadena inicial arbitraria.***

El valor de *hash* del paquete completo será el último valor de *hash* obtenido, esto es, h_{k-1}.

Se reportan varias funciones de *hash* y el análisis de roturas de algunas de ellas.[2-8]

Como quiera que los cifradores de bloques, trabajan con bloques de datos de longitud fija,[9,10] se presume que una buena selección para la función de hash f, es precisamente, la de un cifrador de bloques seguro.

La figura 1, muestra el esquema general de una función de *hash* basada en un cifrador de bloques.

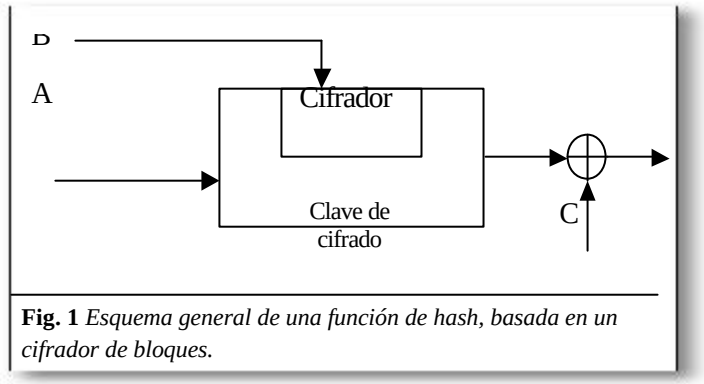

Fig. 1 *Esquema general de una función de hash, basada en un cifrador de bloques.*

El esquema general de estas funciones puede presentarse como:

$$h_i = E_A(B) \oplus C \qquad ...(2)$$

E_A: Algoritmo simétrico E con clave de cifrado A.

donde:

h_0: Valor inicial *t* seleccionado aleatoriamente.

A, *B* y *C*: Pueden ser indistintamente M_i, h_{i-1}, $Mi \oplus h_{i-1}$ o una constante.[2]

La expresión 2 puede generar 64 funciones de *hash* diferentes, muchas de las cuales son criptográficamente débiles. En las cuales pueden encontrarse en la referencia 2.

*DLL: Dinamic Link Library, de su denominación en idioma inglés.

**hash: en idioma inglés, carne molida. Modo sugerente de nombrar una función unidireccional.

**Es presumible que, en algunas implementaciones, mantener en secreto esta cadena que puede hacer más segura la función de *hash*.

Una de estas funciones es la propuesta en las referencias, donde se emplea el cifrador IDEA,[10] que trabaja con un bloque de 128 bit sobre un bloque de 128 bit.

El esquema propuesto es la función:

$$h_i=E_{hi-1,Mi}(h_{i-1}) \quad ...(4)$$

donde:

h_0 es un valor *t* generado aleatoriamente.

Nótese que la clave de 128 bit se obtiene al concatenar los paquetes h_i-1 y M_i, ambos de 64 bit.

Al decidir emplear una función de compresión unidireccional, como mecanismo de control de integridad, se genera el valor de *hash* para el paquete o archivo de datos a proteger.

De esta manera se puede detectar un ataque a la integridad debido a una modificación del paquete, ya sea por un atacante conciente o motivado por un error en el almacenamiento o transmisión.

Dado que no es posible recuperar el paquete original a partir del valor de compresión, se ha producido una falla de integridad si el *hash* recalculado con el paquete analizado no se corresponde con el valor original. Por este motivo, resulta obvio brindar la debida protección al valor de *hash*.

En otro orden de cosas, el desconocimiento del valor de *hash* inicial, que forma parte del algoritmo de compresión empleado, es un elemento de seguridad, si se mantiene en secreto.

CONCLUSIONES

La integridad de los datos, como importante atributo de la seguridad, es un aspecto sistemáticamente olvidado en muchas implementaciones y enfoques, lo cual se observa en los escenarios descritos.

Obviar la incorporación de controles de integridad puede dar al traste con otras medidas de seguridad tomadas por desarrolladores y usuarios, porque el sistema puede ser incapaz de autodetectar ataques a la integridad de sus propios componentes.

Es necesario aplicar algunas nociones clásicas de redundancia que permitan validar la integridad de los paquetes de datos, seleccionando entre las alternativas posibles.

El empleo de las funciones de *hash* criptográficas se anuncian como el artificio más completo para alcanzar controles de integridad robustos y puede incorporar una componente secreta que impida a los atacantes sustituir estos por valores fraudulentos.

RECOMENDACIONES

La aplicación de funciones criptográficas, en general, está sujeta a restricciones nacionales e internacionales e implica un procesamiento adicional, que puede ser costoso en aplicaciones de tiempo real.

De manera general, se deben incrementar los intercambios de criterios entre desarrolladores, académicos y usuarios, acerca de temas de seguridad informática, abordando, en particular la inclusión de controles de integridad.

Para cada desarrollo informático, en particular, debe abordarse la pertinencia concreta de decidir que mecanismo de control de integridad debe ser aplicado.

REFERENCIAS

1.SEPÚLVEDA L., R.: "Protocolo de enlace de datos para la confidencialidad y la autenticidad de las comunicaciones" Tesis doctoral, Ciudad de La Habana, Cuba, 1998.

2.SCHNEIER, B.: *Applied Cryptography Second Edition Protocols, Algorithms, and Source Code in C*, 1996.

3. *http://www.Enginetools.tsx.org.* Consultado el 8 de julio 2003

4. Rainbow Technologies: *Sentinel SuperPro 6.0. Developers´s Guide*, Junio 2000. (Documento pdf distribuido como parte del producto.)

5. COMER. D.C.: *Redes globales de información con Internet y TCP/IP.Principios básicos, protocolos y arquitectura* 3ra. ed., Prentice, Hall, 1996.

6. DMITREV,V.I.: *Teoría de información aplicada*, Ed. MIR Moscú, 1989.

7. LAI, X. AND J. MASSEY: "Hash Functions Based on Block Ciphers", *Advances in Cryptology-EUROCRYPT´92 Proceedings*, Springer- Verlag, pp. 55-70, 1992.

8.PRENEEL, B.: "Análisis and Design of Cryptographic Hash Functions", PhD. Dissertaction, Katholieke Universiteit Leuven, Jan., 1993.

9. DAEMEN, J. AND V. RIJMEN: The Rijndael Block Cipher. AES Proposal. Disponible en: http://www.esat.kuleuven.ac.be/~rijmen/rijndael/.

10. XUEJIA, LAI: "On the Design and Security of Block Ciphers" *ETH Series in Information Processing*, Editor James L. Massey Vol. 1, Hartung-Gore Verlag Konstanz, Zurich, 1992.

Disponemos de un departamento informatizado, dotado con tecnologías que nos permiten realizar todo el proceso de edición de revistas científicas así como de otros materiales.

Visítenos!!!

Departamento de Ediciones-Imprenta
Instituto Superior Politécnico
José Antonio Echeverría, CUJAE
www. cujae.edu.cu/ediciones
http://intranet/ediciones

EDICIONES IMPRENTA CUJAE