Capítulo 1: Introducción a la seguridad informática

Autores: Ing. Parisi Germán - Ing. Bertola Federico

Agradecimientos especiales por sus aportes: Ing. Barrionuevo Ileana, Mangini Agustín.

Repositorio: https://github.com/Gochip/desarrollo-seguro-plataformas-web

Introducción

En la actualidad, la seguridad informática ha ido tomando un papel más relevante durante la vida de cualquier tipo de software debido a que los usuarios están más concientizados y los ciberdelincuentes cada vez están más activos. La misma tiene como objetivo proteger la información a través de medidas preventivas y reactivas. Para ello todo desarrollo de un sistema informático debe ser guiado por un plan de seguridad informática.

Para elaborar ese plan, debemos comprender que la seguridad informática se basa en los siguientes 3 pilares:

1. Confidencialidad de la información.
2. Integridad de la información.
3. Disponibilidad de la información.

¡Leer!
A partir de ahora nos enfocaremos en la seguridad informática en entornos web. Lo que quiere decir que la mayoría de los ejemplos estarán contextualizados en este tipo de plataformas. Sin embargo, se pueden generalizar fácilmente.

Algunos autores podrían agregar a un cuarto y quinto pilar denominados autenticación y no-repudio. En este texto los incluiremos como si fueran parte de los pilares de confidencialidad y de integridad respectivamente.

Confidencialidad

La confidencialidad hace referencia a que la información sea accesible por aquellas personas autorizadas a la misma. Algunas fuentes la definen como “la propiedad que garantiza que la información no sea revelada a personas, procesos o dispositivos no autorizados. La confidencialidad incluye a los datos **en reposo**, durante **su procesamiento** y también **en tránsito**” [https://csrc.nist.gov/glossary/term/confidentiality]. Una de las principales técnicas que se utiliza para lograr esto es la ***criptografía***. Aunque cabe aclarar que, en general, es necesaria pero puede no ser suficiente, es decir, uno puede implementar y aplicar criptografía en un sistema web y que el mismo siga generando información no confidencial. Más adelante en el texto veremos porqué.

Bien, pero ¿qué es la criptografía? Se podría definir de manera muy simple como el arte de ocultar mensajes. Veamos un ejemplo. Supongamos el siguiente mensaje:

```
desarrollo seguro de plataformas web
```

Entonces, podríamos cifrarlo y lograr el siguiente mensaje cifrado:

Autores: Ing. Parisi Germán - Ing. Bertola Federico
Agradecimientos especiales por sus aportes: Ing. Barrionuevo Ileana, Mangini Agustín.

ghvduuroorcvhjxurcghcsodwdirupdvczhe

El algoritmo utilizado anteriormente se denomina algoritmo de César, en honor al emperador romano que fue su inventor. La idea detrás de este algoritmo es bastante simple. Cada letra del mensaje que queremos ocultar (también llamado mensaje en claro) se reemplaza por una letra de N posiciones más adelante en el alfabeto, por ejemplo, si fijamos a N=3 y se necesita cifrar a la letra *e*, la letra que la reemplazará será la *h*, ya que es la que le sigue en 3 posiciones más adelante en el alfabeto, es decir, está la *f*, la *g* y luego la *h*.

e	f	g	h
0	1	2	3

A continuación se presenta el algoritmo de César implementado en Java. Se recomienda detenerse y leer este algoritmo hasta entenderlo.

Algoritmo César implementado en Java

```
public class Cesar {

   public static String alfabeto = "abcdefghijklmnopqrstuvwxyz ";
   public static int movimiento = 3;
   public static void main(String args[]) {
      String mensajeEnClaro = "desarrollo seguro de plataformas web";

      String mensajeCifrado = "";
      for (int i = 0; i < mensajeEnClaro.length(); i++) {
         char c = mensajeEnClaro.charAt(i);
         char caracterCifrado = getCaracterCifrado(c);
         if (caracterCifrado == '\u0000') {
            System.out.println("Error: Se ingresó un caracter que no se
pudo cifrar.");
            return;
         } else {
            mensajeCifrado += caracterCifrado;
         }
      }
      System.out.println(mensajeCifrado);
   }

   public static char getCaracterCifrado(int caracterSinCifrar) {
      for (int i = 0; i < alfabeto.length(); i++) {
         char c = alfabeto.charAt(i);
```

Autores: Ing. Parisi Germán - Ing. Bertola Federico
Agradecimientos especiales por sus aportes: Ing. Barrionuevo Ileana, Mangini Agustín.

Repositorio: https://github.com/Gochip/desarrollo-seguro-plataformas-web

```
            if (caracterSinCifrar == alfabeto.charAt(i)) {
               return alfabeto.charAt((i + movimiento) %
alfabeto.length());
            }
         }
         return '\u0000';
      }
}
```

Ubicado en el repositorio de este libro en: `capitulo-1/confidencialidad/Cesar.java`

Ejercicio I - Capítulo 1

Modificar el anterior algoritmo con lo siguiente:

1. Solicitar por teclado la cantidad de posiciones a moverse para ejecutar el reemplazo. Debe aceptar números enteros negativos indicando un sentido de movimiento opuesto. Por ejemplo, si un número entero positivo indica moverse hacia la derecha, un número entero negativo indicaría moverse hacia la izquierda.
2. El alfabeto debe soportar letras mayúsculas, símbolos de puntuación y vocales con tildes (áéíóú).

Tip: En Java, podés utilizar la clase Scanner para realizar la lectura por teclado.

Recordemos que apenas hemos introducido al atributo de la confidencialidad, inmediatamente después hemos hablado que la criptografía es necesaria pero puede no ser suficiente.

Al margen de esto, vale la pena aclarar que el algoritmo César es **fácilmente vulnerable**, es por eso, que en el Capítulo 5 (Criptografía Aplicada) veremos los algoritmos criptográficos que pueden y deben ser utilizados en sistemas reales. Y, por sobre todo, se recomienda encarecidamente no implementar algoritmos de cifrado propios, siempre es conveniente utilizar aquellas implementaciones que ya fueron hechas y probadas por expertos.

La criptografía no asegura confidencialidad en un sitio web

Es muy probable que a la mayoría que esté leyendo esto, le parezca evidente la afirmación expuesta en el título de esta sección pero a muchos otros no. Intentemos entenderlo con más profundidad. Supongamos que la plataforma web que estamos desarrollando cuenta con un backend con APIs REST. Éstas probablemente necesiten algún mecanismo de autenticación y autorización. El primero hace referencia a determinar si el usuario que quiere acceder es quien dice ser, el segundo hace referencia a si el usuario que ya está autenticado (es decir, el sistema web ya sabe quién es) puede o no acceder a este

recurso. Ahora bien, supongamos que existe una falla en el mecanismo de autenticación o autorización, entonces estaríamos ante una probable fuga de los datos afectando la confidencialidad de los mismos. En los próximos capítulos profundizaremos en estos casos y analizaremos ejemplos.

Integridad

La integridad es la propiedad de la información que asegura que la misma no haya sido alterada por usuarios malintencionados y que el usuario que debe acceder a esa información, lo hace de manera completa y correcta. Es decir, se debe proteger contra modificaciones o destrucción, y esto implica asegurar el no repudio y autenticidad de la información [https://csrc.nist.gov/glossary/term/integrity].

Particularmente lo que nos interesa respecto a la integridad de la información es poder determinar si la misma ha sido alterada. Se han diseñado algoritmos que permiten generar una "marca" de un determinado contenido. Por ejemplo, dado un archivo X, se puede obtener una marca generalmente conocida como hash del archivo y, entonces, si el archivo X es modificado en al menos un bit, ese hash cambiará rotundamente, por lo que podemos asegurar que no se trata del mismo archivo.

Existen varios tipos de algoritmos generadores de hashes (a veces llamados funciones hash). Los dos más conocidos son MD5 y SHA. Particularmente para la segunda existen algunas variantes, por lo que se la suele llamar familia de funciones SHA. Veamos un ejemplo:

Algoritmo SHA-256

```
import java.security.*;
import java.nio.charset.*;
import java.math.*;

public class Sha256 {
   public static void main(String args[]) {
      try {
         MessageDigest generadorHash =
MessageDigest.getInstance("SHA-256");

         String texto = "CORDOBA CAPITAL";

         // Se obtiene el hash en bytes.
         byte[] hashBytes =
generadorHash.digest(texto.getBytes(StandardCharsets.UTF_8));

         // Se imprime el resultado en bytes.
         System.out.println("Resultado en bytes:");
         for (byte i = 0; i < hashBytes.length; i++) {
```

Autores: Ing. Parisi Germán - Ing. Bertola Federico
Agradecimientos especiales por sus aportes: Ing. Barrionuevo Ileana, Mangini Agustín.

```
            byte h = hashBytes[i];
            System.out.println(h);
        }

        // Se convierte el hash en formato hexadecimal.
        System.out.println("Resultado en hexadecimal:");
        BigInteger hashNumero = new BigInteger(1, hashBytes);
        String hashString = hashNumero.toString(16);
        System.out.println(hashString);
      } catch (NoSuchAlgorithmException ex) {

      }
   }
}
```

Ubicado en el repositorio de este libro en: `capitulo-1/integridad/Sha256.java`

El anterior programa muestra el hash SHA-256 de la cadena `"CORDOBA CAPITAL"` en una secuencia de bytes y, luego, para una visualización más estándar se lo convierte a hexadecimal. Tener en cuenta que los hashes normalmente se representan en hexadecimal.

Solo para corroborar, ahora generaremos el hash SHA-256 del string anterior a través de una línea de comandos en Linux:

```
echo -n "CORDOBA CAPITAL" | sha256sum
```

Revisar que la salida de esta línea de comandos es igual a la generada por la del programa Java.

Como ejercitación adicional vamos a generar el hash SHA-256 de un archivo a partir de Powershell (la interfaz de consola de Windows). También se presenta el comando para hacerlo en Linux.

Windows: Generación de hash en Powershell

```
PS C:\Users\german> Get-FileHash .\Sha256.java
```

La salida que se obtiene al ejecutar este comando es el hash que “representa” al contenido del `Sha256.java`. En caso que se modifique un bit del archivo, el hash será otro completamente distinto al anterior, es por esto que se utilizan las funciones hash para validar integridad.

Linux: Generación de hash en Bash

```
$ sha256sum .\Sha256.java
```

Autores: Ing. Parisi Germán - Ing. Bertola Federico
Agradecimientos especiales por sus aportes: Ing. Barrionuevo Ileana, Mangini Agustín.

Muchas veces, cuando uno descarga un archivo de internet a través de HTTP, nos brindan un hash (MD5 o SHA) para poder corroborar que se descargó bien. Justamente con esto validamos la integridad del archivo descargado. Esta misma idea se aplica a sistemas web y es conocido como Subresource Integrity (SRI). Básicamente se agrega un hash de verificación de integridad a las etiquetas script o link y, en caso que el navegador web verifique que el hash no corresponda al contenido descargado, directamente no lo ejecuta. Veremos más sobre SRI en el capítulo 4.

No repudio

Es una característica de seguridad que indica que si un usuario modifica determinada información, ese usuario no puede alegar que no fue él quién la modificó. Dicho de otra manera, si un usuario modifica cierto dato, efectivamente fue ese usuario y no pudo haber sido otro, y tampoco tiene manera de denegarlo. Por ejemplo, las firmas digitales es uno de los mejores métodos para implementar el no repudio.

Disponibilidad

La disponibilidad es un atributo de la información que implica que la misma esté a disposición cuando las personas autorizadas lo requieran. También se suele referir a esta propiedad como aquella que asegura el uso confiable y oportuno de la información [https://csrc.nist.gov/glossary/term/availability]. Como es evidente, es difícil obtener un nivel de disponibilidad del 100% en un sistema web. Esto depende tanto de la aplicación como así también de la infraestructura en la cual esté montada. En esta infraestructura, existen muchos componentes que pueden fallar tanto por un error sin intención como así también a causa de algún tercero que lo provocó. En cualquier caso, es necesario definir un plan de reacción para poder afrontar estos eventos inesperados.

Para definir un plan realista, es necesario comprender correcta y completamente los siguientes aspectos:

1. Arquitectura del software.
2. Arquitectura de la infraestructura.
3. Servicios externos.

Procederemos a profundizar en los 3 anteriores puntos.

Arquitectura del software

Es necesario identificar servicios web, páginas, controladores, modelos, acceso a base de datos, transferencia de archivos al cliente, etc. Dependiendo de la arquitectura del sistema web, se deberán plantear acciones para actuar ante un evento no deseado. En general, los puntos claves de la disponibilidad en un sistema web son: acceso a disco (normalmente a una base de datos) y el uso de la red en las peticiones (tanto en recibir como en enviar información hacia el front o hacia otros sistemas).

Una arquitectura basada en microservicios y en código asincrónico suele aumentar el nivel de disponibilidad e incluso permite ser más resistentes ante ataques de denegación de servicio.

Arquitectura de la infraestructura

En este punto es necesario identificar servidores web, servidores proxy, firewall, herramientas para monitoreo, servidores de bases de datos, sistemas operativos, etc.

Básicamente un sistema web puede ser desplegado en una infraestructura de hardware propietaria, en donde, es muy probable que se requieran configuraciones para fortalecer esta infraestructura. Otra alternativa es usar servicios de hosting, o bien, servicios en la nube que normalmente incluyen configuraciones que mejoran la disponibilidad. Incluso, es posible diseñar un esquema híbrido para fortalecer su disponibilidad.

Servicios externos

Si el sistema web se comunica con otros sistemas, a través de, por ejemplo, una petición HTTP, es importante actuar ante aquellos casos en que este servicio tercero no funcione como esperamos.

Seguridad en el ciclo de desarrollo de software: Flaw vs bug

El desarrollo de software avanza a través de diferentes etapas, independientemente de la metodología que se use, y desde el primer momento se debe tener en cuenta la seguridad. A esto se le suele llamar, seguridad desde los requerimientos. Como sabemos, un software va a tener errores pero depende en qué momento se genera ese error, es cuándo se denomina Flaw o Bug.

Un flaw es un error de diseño y suele ser muy costoso arreglarlo ya que en general requiere muchos cambios. Un bug es un error de código y, en general, debería ser más barato arreglarlo.

Por ejemplo, si la identificación de un recurso determinado del backend como la tabla clientes, se realiza mediante enteros auto-incrementales y desde el frontend es referenciada así, esto puede generar que un ciberdelincuente pueda enumerar a todos ellos. Es decir, la siguiente tabla muestra un ejemplo de estas referencias:

Cliente 1	`/api/clientes?id=1`
Cliente 2	`/api/clientes?id=2`
Cliente N	`/api/clientes?id=N`

Este problema planteado es un flaw porque durante el diseño no se pensó que este campo podría generar este tipo de inconvenientes sobre la confidencialidad. Como se puede notar, resolver este flaw requiere cambios en la base de datos y en cada capa de la aplicación web para que ahora se referencie a un cliente usando otro tipo de identificador y no usando un entero auto-incremental.

Autores: Ing. Parisi Germán - Ing. Bertola Federico
Agradecimientos especiales por sus aportes: Ing. Barrionuevo Ileana, Mangini Agustín.

Por otro lado, si un ciberdelincuente puede enumerar a todos los clientes mediante un ataque de SQLinjection, lo más probable es que estemos ante un bug de código y se haya utilizado una mala práctica de código.

Lamentablemente, desde el punto de vista de seguridad, tanto un flaw como un bug, pueden generar vulnerabilidades muy críticas.

Guía de ejercicios

1. Modificar el algoritmo de cifrado César escrito en Java con lo siguiente:
    a. Solicitar por teclado la cantidad de posiciones a moverse para ejecutar el reemplazo. Debe aceptar números enteros negativos indicando un sentido de movimiento opuesto. Por ejemplo, si un número entero positivo indica moverse hacia la derecha, un número entero negativo indicaría moverse hacia la izquierda.
    b. El alfabeto debe soportar letras mayúsculas, símbolos de puntuación y vocales con tildes (áéíóú).
2. Investigue los métodos actuales para transmitir datos de forma confidencial a través de internet. ¿Qué protocolos se utilizan? ¿Qué algoritmos criptográficos se utilizan?
3. La verificación de redundancia cíclica (CRC) es un método para detectar errores de integridad en una serie de datos. ¿En dónde se puede aplicar? ¿Serviría para la detección de integridad en archivos normalmente utilizados?
4. En el sistema tributario argentino existe una clave llamada CUIT que contiene un dígito verificador. ¿Para qué sirve ese dígito?
5. ¿Qué es una botnet? ¿En qué pilar fundamental de la seguridad informática puede afectar? ¿Qué es un ataque DDoS? ¿Cómo podría implementar uno?
6. En numerosas plataformas web es necesario iniciar sesión mediante un usuario y una clave, ¿considera a este esquema lo suficientemente seguro?
7. Luego de una prueba de seguridad, se encontró que a través de un parámetro de una API REST se puede ejecutar código arbitrario del sistema operativo. Se estima que este error viene por una incorrecta validación de ese parámetro ¿Estamos ante un flaw o un bug?
8. Luego de una prueba de seguridad a un home banking, se encontró que un ciberdelincuente podría tener acceso a todos los comprobantes de las transacciones realizadas por cualquier usuario. Además de un problema obvio de validación de permisos, se percató que los comprobantes están identificados por un número entero autoincremental, lo que aún facilita más la iteración por todos los comprobantes. En este caso, ¿estamos ante un flaw o bug?

Autores: Ing. Parisi Germán - Ing. Bertola Federico
Agradecimientos especiales por sus aportes: Ing. Barrionuevo Ileana, Mangini Agustín.

Repositorio: https://github.com/Gochip/desarrollo-seguro-plataformas-web

Anexos del capítulo 1

Anexo 1: Implementación del algoritmo César en Python

Algoritmo César implementado en Python

```python
class Cesar:
    ALFABETO = "abcdefghijklmnñopqrstuvwxyz "

    def __init__(self, movimiento=3):
        self.movimiento = movimiento

    def cifrar(self, mensaje):
        mensaje_cifrado = ""
        for car in mensaje:
            mensaje_cifrado += self._get_caracter_cifrado(car)

        return mensaje_cifrado

    def _get_caracter_cifrado(self, car_no_cifrado):
        i_nc = self.ALFABETO.find(car_no_cifrado)

        if i_nc < 0:
            raise Exception(f"Caracter inválido: {car_no_cifrado}")

        i_cf = (i_nc + self.movimiento) % len(self.ALFABETO)
        caracter_cifrado = self.ALFABETO[i_cf]

        return caracter_cifrado
```

Ubicado en el repositorio de este libro en: `capitulo-1/confidencialidad/cesar.py`

Autores: Ing. Parisi Germán - Ing. Bertola Federico
Agradecimientos especiales por sus aportes: Ing. Barrionuevo Ileana, Mangini Agustín.

Anexo 2: Algoritmo MD5

El objetivo del algoritmo MD5 es generar un string de 32 hexadecimales dado un string de N caracteres que se componen de T bits dónde T < 2 ^ 64. A este string de N caracteres le llamaremos ***mensaje***.

Pasos que ejecuta el algoritmo

PASO 1: Adición de bits

1. Convierte el string recibido a una representación ASCII.
2. La representación en ASCII es dividida en bloques de 512 bits. Evidentemente, el último bloque puede no tener los 512 bits completos y habrá que completarlos con un padding. Pero este padding no debe completarse hasta los 512 bits sino que hasta los 448 bits, ya que los últimos 64 bits se reservan para especificar el tamaño original del mensaje en bits (es decir, sin el padding). Es por eso, que como se reservan 64 bits para el tamaño, entonces el string máximo de entrada debería tener 2 ^ 64 bits.

Ahora bien, para completar el padding en el último bloque. Acá hay dos opciones que pueden ocurrir:

CASO 1: El último bloque tiene menos de 448 bits: Se debe aplicar un padding hasta completar los 448 bits (un 1 y el resto de los bits en 0).

CASO 2: El último bloque tiene 448 bits o más: Se debe aplicar un padding hasta completar los 512 bits de este último bloque y los primeros 448 bits de un bloque extra que se debe agregar después del último.

PASO 2: Agregar longitud del mensaje

Tanto en el último bloque del caso 1 como en el bloque añadido del caso 2, se debe establecer el tamaño del mensaje original (sin el padding) en los últimos 64 bits.

En los pasos 1 y 2, se asegura que el tamaño del mensaje de T bytes será extendido hasta que sea congruente con 448, módulo 512.

Para entender mejor estos 2 primeros pasos, se puede hacer uso del siguiente simulador:

Simulador del algoritmo MD5

Autores: Ing. Parisi Germán - Ing. Bertola Federico
Agradecimientos especiales por sus aportes: Ing. Barrionuevo Ileana, Mangini Agustín.

Repositorio: https://github.com/Gochip/desarrollo-seguro-plataformas-web

https://fthb321.github.io/MD5-Hash/MD5OurVersion2.html

A partir de este punto se estructuró el mensaje de entrada y se lo dejó listo para empezar a calcular el hash.

PASO 3: Inicializar el búfer MD

Se inicializan 4 variables (también llamado vector de inicialización) para calcular el hash. Los valores más comunes en hexadecimal son (aunque podrían ser otros):

```
A = 0x67452301
B = 0xEFCDAB89
C = 0x98BADCFE
D = 0x10325476
```

PASO 4: Procesado de mensaje en bloques de 16 palabras

Se divide al mensaje en bloques de 16 palabras, dónde cada palabra contiene 32 bits. Las 4 variables inicializadas anteriormente se van operando bit a bit con los bloques del mensaje a través de 4 funciones que se suelen llamar F, G, H, I. Para más detalles remitirse al RFC 1321.

PASO 5: Salida

Finalmente, las 4 variables que se inicializaron, se van modificando junto con el mensaje hasta que se obtiene el resultado. El mismo está dado por el byte de menor peso de la variable A hasta el byte de mayor peso de la variable D, pasando por B y C. Es decir:

A	**BYTE1A** | BYTE2A | BYTE3A | BYTE4A
B	BYTE1B | **BYTE2B** | BYTE3B | BYTE4B
C	BYTE1C | BYTE2C | **BYTE3C** | BYTE4C
D	BYTE1D | BYTE2D | BYTE3D | **BYTE4D**
Hash MD5	**BYTE1A | BYTE2B | BYTE3C | BYTE4D**

RFC 1321: The MD5 Message-Digest Algorithm

https://www.ietf.org/rfc/rfc1321.txt

Autores: Ing. Parisi Germán - Ing. Bertola Federico
Agradecimientos especiales por sus aportes: Ing. Barrionuevo Ileana, Mangini Agustín.

Aclaración

El algoritmo MD5 fue roto en varias ocasiones a lo largo del tiempo y por lo tanto no se recomienda usarlo a menos que se sepa lo que se está haciendo. En algunas ocasiones, el algoritmo todavía sigue siendo útil y perfectamente aplicable. En el capítulo sobre criptografía aplicada profundizaremos sobre esto.

Autores: Ing. Parisi Germán - Ing. Bertola Federico
Agradecimientos especiales por sus aportes: Ing. Barrionuevo Ileana, Mangini Agustín.