Laboratorio 6
Seguridad de una Base de Datos

Objetivo:
El alumno aprenderá a aplicar los conceptos de seguridad a la Base de Datos.

Temas de la materia abordados, segun el plan de estudios:
UNIDAD V.- SEGURIDAD Y AUDITORÍA DE UNA BASE DE DATOS

TIEMPO APROXIMADO: 10 Horas

OBJETIVO DE LA UNIDAD: Entender y aplicar los conceptos para mantener la seguridad y auditar una base de datos, a través de diversas tareas programadas que facilitan el mantener la seguridad, integridad control de accesos concurrentes, transacciones y monitoreo de la base de datos.

CONTENIDO
5.1 Problemas de seguridad en una base de datos
    5.1.1 Aspectos fundamentales de seguridad
    5.1.2 Amenazas a una base de datos
5.2 Medidas de seguridad en una base de datos
    5.2.1 Mecanismos de seguridad
    5.2.2 Autorizaciones y vistas
    5.2.3 Cifrado

Partes de la que se compone este laboratorio:
● Introducción.
● Conceptos básicos
● Creación de programas y problemas.
● Referencias.

Introducción.
Una vez que ha concluido el diseño y configuración de la base de datos es necesario que que desarrolle las aplicaciones (usando lenguaje de programación Java) desde las cuales se les dará mantenimiento a las tablas, debe recordar que solo se permite a ciertos usuarios acceder a algunas tablas, mantenga estas restricciones a nivel de la aplicación. Para ingresar al sistema los usuarios deben de capturar login y password, mismo que se usará para conectarse a la base de datos. Modifique las tablas para que en todos los casos se almacene, sin que el usuario lo sepa, login, fecha y hora de cada cambio a los registros, esto por cuestión de seguridad. Cada una de las ventanas dará acceso, de acuerdo al perfil del usuario, es decir, habrá ventanas que solo se permiten acceder al personal del departamento Escolar, mientras que otras solo a los del departamento Administrativo y algunas otras a ambos.

A continuación se detalla el problema con el que se trabajará en este laboratorio:

Proyecto Universidad ACME
En UACME, se ofrecen dos tipos de cursos en el periodo especial de verano, en el cual se

imparten cursos de verano y cursos extracurriculares. Los primeros son materias que un alumno regular que estudia una carrera cursa en este periodo, se le permite adelantar hasta dos materias; mientras que los segundos son cursos especiales de capacitación que se ofrecen a alumnos regulares como estudiantes o profesionistas externos.
Los docentes de la UACME, son los únicos a los que se les permite impartir estos cursos, por los cuales recibe un pago adicional, se les paga de acuerdo a un tabulador que indica el costo de la hora de estos cursos de acuerdo al nivel académico del docente. El pago se genera a partir de la alta del curso y solo se permite expedir un cheque por cada curso. Además los estudiantes deben acudir a pagar adicionalmente al costo del semestre por asistir a ellos. UACME tiene dos departamentos que intervienen en la administración de los cursos: A)Departamento de Administración (DA) y B)Departamento de Control Escolar (DCE). Corresponde al DA, efectuar el pago a los docentes y los cobros a los alumnos. El DA es dirigido por el C.P. Ávila y es auxiliado por el Sr. Cancino. Mientras que el DCE, es dirigido por el Lic. Barroso y auxiliado por los Sras. Tirado, Martínez, Aquino y Ramos y es en este donde se decide cuales cursos se imparten en el periodo, quién los imparte, y se aceptan las solicitudes de los alumnos. Un caso especial, es el de los Profesores, ya que el DA es quién les puede modificar el sueldo quincenal, mientras que el DCE ni siquiera puede visualizar éste. Lo curioso radica en que, es el DCE quién acepta los docentes y los registra en el sistema, pero es el DA donde se captura el sueldo. Importante es para la administración de la UACME que esta política se aplique al pie de la letra, y que sea implementado directamente sobre la DB.

Conceptos básicos.

Definición de autenticación o autentificación.

En términos de seguridad de redes de datos, se puede considerar uno de los tres pasos fundamentales (AAA). Cada uno de ellos es, de forma ordenada:

1. Autenticación En la seguridad de ordenador, la autenticación es el proceso de intento de verificar la identidad digital del remitente de una comunicación como una petición para conectarse. El remitente siendo autenticado puede ser una persona que usa un ordenador, un ordenador por sí mismo o un programa del ordenador. En un web de confianza, "autenticación" es un modo de asegurar que los usuarios son quién ellos dicen que ellos son - que el usuario que intenta realizar funciones en un sistema es de hecho el usuario que tiene la autorización para hacer así.
2. Autorización Proceso por el cual la red de datos autoriza al usuario identificado a acceder a determinados recursos de la misma.
3. Auditoría Mediante la cual la red o sistemas asociados registran todos y cada uno de los accesos a los recursos que realiza el usuario autorizados o no.

El problema de la autorización a menudo, es idéntico a la de autenticación; muchos protocolos de seguridad extensamente adoptados estándar, regulaciones obligatorias, y hasta estatutos están basados en esta asunción. Sin embargo, el uso más exacto describe la autenticación como el proceso de verificar la identidad de una persona, mientras la autorización es el proceso de verificación que una persona conocida tiene la autoridad para realizar una cierta operación. La autenticación, por lo tanto, debe preceder la autorización. Para distinguir la autenticación de la autorización de término

estrechamente relacionada, existen unas notaciones de taquigrafía que son: A1 para la autenticación y A2 para la autorización que de vez en cuando son usadas, también existen los términos AuthN y AuthZ que son usados en algunas comunidades.

Métodos de autenticación

Los métodos de autenticación están en función de lo que utilizan para la verificación y estos se dividen en tres categorías:

- Sistemas basados en algo conocido. Ejemplo, un *password* (Unix) o *passphrase* (PGP).
- Sistemas basados en algo poseído. Ejemplo, una tarjeta de identidad, una tarjeta inteligente(*smartcard*), dispositivo usb tipo epass token, smartcard o dongle criptográfico.
- Sistemas basados en una característica física del usuario o un acto involuntario del mismo: Ejemplo, verificación de voz, de escritura, de huellas, de patrones oculares.

Cualquier sistema de identificación ha de poseer unas determinadas características para ser viable:

- Ha de ser fiable con una probabilidad muy elevada (podemos hablar de tasas de fallo de en los sistemas menos seguros).
- Económicamente factible para la organización (si su precio es superior al valor de lo que se intenta proteger, tenemos un sistema incorrecto).
- Soportar con éxito cierto tipo de ataques.
- Ser aceptable para los usuarios, que serán al fin y al cabo quienes lo utilicen.

Control de acceso.

Un ejemplo familiar es el control de acceso. Un sistema informático supuesto para ser utilizado solamente por aquellos autorizados, debe procurar detectar y excluir el desautorizado. El acceso a él por lo tanto es controlado generalmente insistiendo en un procedimiento de la autenticación para establecer con un cierto grado establecido de confianza la identidad del usuario, por lo tanto concediendo esos privilegios como puede ser autorizado a esa identidad. Los ejemplos comunes del control de acceso que implican la autenticación incluyen:

- Retirar de dinero de un cajero automático.
- Control de un computador remoto sin Internet.
- Uso de un sistema Internet bancario.

Sin embargo, observar que mucha de la discusión sobre estos asuntos es engañosa porque los términos se utilizan sin la precisión. Parte de esta confusión puede ser debido “al tono de la aplicación de ley” de mucha de la discusión. Ninguna computadora, programa de computadora, o poder del usuario de la computadora “confirman la identidad” de otro partido. No es posible “establece” o “probar” una identidad, cualquiera. Hay ediciones difíciles que están al acecho debajo de qué aparece ser una superficie directa.

Es solamente posible aplicar una o más pruebas que, si están pasadas, se han declarado

previamente para ser suficientes proceder. El problema es determinarse qué pruebas son suficientes, y muchos tales son inadecuadas. Tienen sido muchos casos de tales pruebas que son spoofed con éxito; tienen por su falta demostrada, ineludible, ser inadecuadas. Mucha gente continúa mirando las pruebas -- y la decisión para mirar éxito en pasar -como aceptable, y para culpar su falta en “sloppiness” o “incompetencia” de parte alguien. El problema es que la prueba fue supuesta para trabajar en la práctica -- no bajo condiciones ideales de ningún sloppiness o incompetencia-y no. Es la prueba que ha fallado en tales casos. Considerar la caja muy común de un email de la confirmación a el cual deba ser contestado para activar una cuenta en línea de una cierta clase. Puesto que el email se puede arreglar fácilmente para ir a o para venir de direcciones falsas y untraceable, éste es justo sobre la menos autenticación robusta posible. El éxito en pasar esta prueba significa poco, sin consideración alguna hacia sloppiness o incompetencia.

Sistemas basados en algo conocido: contraseñas

El modelo de autenticación más básico consiste en decidir si un usuario es quien dice ser simplemente basándonos en una prueba de conocimiento que *a priori* sólo ese usuario puede superar; y desde Alí Babá y su *`Ábrete, Sésamo'* hasta los más modernos sistemas Unix, esa prueba de conocimiento no es más que una contraseña que en principio es secreta. Evidentemente, esta aproximación es la más vulnerable a todo tipo de ataques, pero también la más barata, por lo que se convierte en la técnica más utilizada en entornos que no precisan de una alta seguridad, como es el caso de los sistemas Unix en redes normales (y en general en todos los sistemas operativos en redes de seguridad media-baja); otros entornos en los que se suele aplicar este modelo de autenticación son las aplicaciones que requieren de alguna identificación de usuarios, como el *software* de cifrado PGP o el escáner de seguridad NESSUS. También se utiliza como complemento a otros mecanismos de autenticación, por ejemplo en el caso del Número de Identificación Personal (PIN) a la hora de utilizar cajeros automáticos.

En todos los esquemas de autenticación basados en contraseñas se cumple el mismo protocolo: las entidades (generalmente dos) que participan en la autenticación acuerdan una clave, clave que han de mantener en secreto si desean que la autenticación sea fiable. Cuando una de las partes desea autenticarse ante otra se limita a mostrarle su conocimiento de esa clave común, y si ésta es correcta se otorga el acceso a un recurso. Lo habitual es que existan unos roles preestablecidos, con una entidad activa que desea autenticarse y otra pasiva que admite o rechaza a la anterior (en el modelo del acceso a sistemas Unix, tenemos al usuario y al sistema que le permite o niega la entrada).

Como hemos dicho, este esquema es muy frágil: basta con que una de las partes no mantenga la contraseña en secreto para que toda la seguridad del modelo se pierda; por ejemplo, si el usuario de una máquina Unix comparte su clave con un tercero, o si ese tercero consigue leerla y rompe su cifrado (por ejemplo, como veremos luego, mediante un ataque de diccionario), automáticamente esa persona puede autenticarse ante el sistema con éxito con la identidad de un usuario que no le corresponde.

Inyección de SQL.

Consiste en la modificación del comportamiento de nuestras consultas mediante la introducción de parámetros no deseados en los campos a los que tiene acceso el usuario. La cual es un tipo de error que puede permitir a usuarios malintencionados acceder a

datos a los que de otro modo no tendrían acceso y, en el peor de los casos, modificar el comportamiento de nuestras aplicaciones.

La Inyección Directa de Comandos SQL es una técnica en la cual un atacante crea o altera comandos SQL existentes para exponer datos escondidos, o sobrescribir datos críticos, o incluso ejecutar comandos del sistema peligrosos en la máquina en donde se encuentra la base de datos. Esto se consigue cuando la aplicación toma información de entrada del usuario y la combina con parámetros estáticos para construir una consulta SQL.

Una inyección SQL sucede cuando se inserta o "inyecta" un código SQL "invasor" dentro de otro código SQL para alterar su funcionamiento normal, y hacer que se ejecute maliciosamente el código "invasor" en la base de datos. La inyección SQL es un problema de seguridad informática que debe ser tomado en cuenta por el programador para prevenirlo. Un programa hecho con descuido o con ignorancia sobre el problema, podrá ser vulnerable y la seguridad del sistema puede quedar ciertamente comprometida. Esto puede suceder tanto en programas corriendo en computadores de escritorio, como en páginas Web, ya que éstas pueden funcionar mediante programas ejecutándose en el servidor que las aloja.

La vulnerabilidad puede ocurrir cuando un programa "arma" descuidadamente una sentencia SQL, con parámetros dados por el usuario, para luego hacer una consulta a una base de datos. Dentro de los parámetros dados por el usuario podría venir el código SQL inyectado. Al ejecutarse esa consulta por la base de datos, el código SQL inyectado también se ejecutará y podría hacer un sinnúmero de cosas, como insertar registros, modificar o eliminar datos, autorizar accesos e, incluso, ejecutar código malicioso en el computador.

Entre las vulnerabilidades habituales que hacen que el código de acceso a datos sea susceptible de ataques de inyección SQL se incluyen:

- Validación de entradas débil.
- Construcción dinámica de instrucciones SQL sin utilizar parámetros de tipo seguro.
- Uso de inicios de sesión en bases de datos con privilegios aumentados.

Nota: Las medidas de seguridad convencionales, como el uso de Secure Socket Layer (SSL) e IP Security (IPSec), no protegen a la aplicación de ataques de inyección SQL.

Para evitar los ataques de inyección SQL, es necesario:

- *Restringir y sanear los datos de entrada. Para comprobar los datos que se sabe que son buenos, hay que validar el tipo, la longitud, el formato y el intervalo.*
- *Utilizar parámetros SQL de tipo seguro para el acceso a los datos. Puede utilizar estos parámetros con procedimientos almacenados o cadenas de comandos SQL construidas dinámicamente.*
- *Utilizar una cuenta que disponga de permisos restringidos en la base de datos. Lo ideal sería conceder permisos de ejecución sólo a procedimientos almacenados seleccionados de la base de datos y no ofrecer acceso directo a tablas.*
- *Evitar que se revele información de errores de la base de datos. En caso de que se produzcan errores en la base de datos, asegúrese de que no se revelen al usuario*

mensajes de error detallados.

Autenticación de usuarios en PostgreSQL.

PostgreSQL ofrece múltiples y diferentes métodos de autentificación. El método usado para autentificar a una conexión de cliente en particular puede ser seleccionada en la base de dirección ip del cliente, base de datos o usuario.

Los nombres de usuario de PostgreSQL están separados lógicamente de los nombres de usuario del sistema operativo sobre el cual el servidor se ejecuta. Si los usuarios de un servidor en particular también tienen cuentas en el sistema operativo del equipo, tiene sentido asignar nombres de usuario de base de datos que coincidan con los nombres de usuario de sistema operativo. Sin embargo, un servidor que acepta conexiones remotas podría tener muchos usuarios de bases de datos que no tienen cuentas en el sistema operativo local, y en tales casos no existe una conexión entre los nombres de usuario de la base de datos y el sistema operativo.

Cuando una aplicación cliente se conecta con el servidor de base de datos, esta especifica cual es el nombre de usuario con el que quiere conectarse, del mismo modo que el usuario que desea conectarse a un sistema Unix como un usuario particular. Dentro del ambiente SQL el nombre de usuario determina los privilegios de acceso a los objetos de la base de datos. De allí que sea esencial restringir los usuarios que puedan conectarse al sistema.

El archivo pg_hba.conf de PostgreSQL.

La autenticación de clientes es controlada por un archivo de configuración, al cual tradicionalmente se le conoce como pg_hba.conf y es almacenado en los directorios de la base de datos. (HBA son las siglas en Inglés para autenticación basada en host). Se instala un archivo pg_hba.conf cuando el directorio de datos es inicializado por initdb. Sin embargo, es posible poner el archivo de configuración de la autenticación en cualquier lugar.

El formato general del archivo pg_hba.conf es un conjunto de registros, uno por linea. Las lineas en blanco son ignoradas, así como aquellas que inician con el carácter #. Un registro se compone de un cierto número de campos los cuales están separados por espacios y/o tabuladores. Los campos pueden contener espacios en blanco y los registros no son multilínea.

Cada registro especifica un tipo de conexión, un rango de direcciones IP de cliente (si acaso es relevante para el tipo de conexión), un nombre de base de datos, un nombre de usuario, y el método de autenticación a ser usado para aquellas conexiones que coincidan con estos parámetros. El primer registro con: un tipo de conexión coincidente, dirección del cliente, base de datos solicitada y nombre de usuario, es usado para efectuar la autenticación. No se leen todos los registro, si uno es elegido y la autenticación falla, los registros subsecuentes no son considerados. Si ningún registro coincide el acceso es denegado.

Un registro puede tener uno de los siete formatos siguientes:

local base-de-datos *usuario método-aut* [*opciones-aut*]
host *base-de-datos usuario* dirección-*CIDR método-aut* [*opciones-aut*]

hostssl *base-de-datos usuario dirección-CIDR método-aut* [*opciones-aut*]
hostnossl *base-de-datos usuario dirección-CIDR método-aut* [*opciones-aut*]
host *base-de-datos usuario* dirección-*IP IP-mask método-aut* [*opciones-aut*]
hostssl *base-de-datos usuario dirección-IP IP-mask método-aut* [*opciones-aut*]
hostnossl *base-de-datos usuario dirección-IP IP-mask método-aut* [*opciones-aut*]

El siginificado de estos campos es el siguiente:

local

Este registro coincide con los intentos de conexión usando sockets de dominio-Unix. Sin un registro de este tipo, las conexiones de sockets de dominio-Unix son desactivadas.

host

Este registro coincide con los intentos de conexión usando TCP/IP. Los registros host coinciden con los intentos de conexiones SSL o non-SSL.

Nota: Las conexiones TCP/IP remotas no serán posibles a menos que el servidor sea arrancado con un valor apropiado para el parámetro de configuración listen_addreses, puesto que el comportamiento por omisión es el de escuchar conexiones TCP/IP solo para el rango de direcciones locales.

hostssl

Este registro coincide con los intentos de conexión usando TCP/IP, pero solo cuando la conexión es efectuada con escriptado SSL.

Para hacer uso de esta opción el servidor debe estar construido con soporte SSL. Por lo que el servidor debe ser habilitado al momento del arranque configurando el el parámetro SSL (ver la Sección 17.8 del manual oficial de PostgreSQL para maypor información).

hostnossl

Este registro tiene una lógica opuesta a hostssl: solo coincide con intentos de conexión efectuados sobre TCP/IP que no usen SSL.

database

Especifican los nombres de bases de datos con los que coincide este registro. El valor ***all*** especifica que aplica para todas las bases de datos. El valor sameuser especifica que el registro coincide si la base de datos solicitada tiene el mismo nombre que el usuario solicita. El valor samerole especifica que el usuario solicitado debe ser un miembro del grupo con el mismo nombre que el de la base de datos. De otra manera, este debe ser el nombre especifico de una base de datos de PostgreSQL. Múltiples nombres de bases de datos pueden ser proporcionados

siempre y cuando sean separados con comas. Un archivo separado que contenga los nombres de las bases de datos puede ser especificado siempre y cuando al nombre le precede el símbolo @.

user

Especifica los nombres de usuarios de la BD coinciden con este registro. El valor all especifica que coincide con todos los usuarios. De otro modo, esto es, ya sea el nombre de algún usuario de base de datos especifico, o un nombre de grupo precedido por +. Múltiples nombres de usuario pueden capturarse con solo separarlos con comas. Se puede especificar un nombre de archivo que contenga los nombres de usuario, al cual debe preceder el símbolo @.

CIDR-address

Especifica el rango de direcciones IP que coinciden con este registro. Este campo contiene una dirección IP en una notación decimal estándar y una longitud de máscara CIDR. La longitud de máscara indica el número de bits de mayor orden de la dirección IP del cliente que debe coincidir. Los bits a la derecha de esta deben ser cero en la dirección IP dada. No debe de haber espacios en blanco entre la dirección IP, la /, y la longitud de máscara CIDR.

Ejemplo típicos de direcciones CIDR son 172.20.143.89/32 para un solo equipo, o 172.20.143.0/24 para un red pequeña, o 10.6.0.0/16 para una más grande. Para especificar un solo equipo, use una máscara CIDR de 32 bits para Ipv4 o de 128 para una Ipv6. En una dirección de red, no debe omitir los ceros del final.

Estos campos solo aplican para registros host, hostssl, y hostnossl.

IP-mask

Estos campos pueden ser usados como una alternativa a la notación de direcciones CIDR. En lugar de especificar la longitud de la máscara, esta se especifica en una columna separada. Por ejemplo, representa una longitud de máscara 8 para una CIDR Ipv4, y 255.255.255.255 representa una longitud de máscara de 32 para una CIDR.

Estos campos solo aplican para registros host, hostssl, y hostnossl.

auth-method

Especifica el método de autenticación a usar cuando una conexión coincide con este registro. A continuación se da un resumen de las opciones posibles, para detalles adicionales busque en la sección 19.3 de la documentación de PostgreSQL en la página www.postgresql.org.

trust

Permitir la conexión de forma incondicional. Este método permite que cualquiera se conecte al servidor de base de datos PostgreSQL usando el nombre de usuario que gusten, sin necesidad de un password.

reject

Rechaza la conexión incondicionalmente. Esto es útil para filtrar ciertos equipos de un grupo.

md5

Requiere que el cliente provea de una clave secreta encriptada en MD5 para la autenticación.

password

Requiere que el cliente provea de una clave secreta NO encriptada, Puesto que la clave secreta es enviada en texto abierto en la red, este mecanismo no debería ser usado en redes NO seguras.

gss

Use GSSAPI para autenticar al usuario. Esto solo esta disponible para conexiones TCP/IP.

sspi

Use SSPI para autenticar al usuario. Esto solo esta disponible para Windows.

krb5

Use Kerberos V5 para autenticar al usuario. Esto solo esta disponible para conexiones TCP/IP.

ident

Obtiene el nombre de usuario en el sistema operativo del cliente (para conexiones TCP/IP, lo hace contactando al servidor ident en el cliente, para conexiones locales lo obtiene del sistema operativo) y verifica que coincida con el nombre de usuario en la base de datos.

ldap

Autentica usando un servidor LDAP.

cert

Autentica usando certificados de cliente SSL.

pam

Autentica usando los modulo enchufables de autenticación (Pluggable Authentication Modules - PAM) servicio otorgado por el sistema operativo.

auth-options

Después del campo *auth-method,* puede haber campos con la forma *name=value* que especifican opciones para el método de autenticación.

Puesto que los registro del archivo pg_hba.conf son examinados secuencialmente para cada intento de conexión, el orden de los registros es significativo. Tipicamente, los primeros registros deben tener parámetros de conexión estrictos y métodos de autenticación mas débiles. Por ejemplo, podría desear usar autenticación confiable para conexiones TCP/IP locales pero requerir un password para las conexiones TCP/IP remotas. En este caso un registro que especifica una autenticación confiable para conexiones desde 127.0.0.1 podrían aparecer antes que un registro que especifica una autenticación de password para un rango más amplio de direcciones IP cliente a las que se les permite el acceso.

El archivo pg_hba.conf es leído al momento del arranque y cuando el servidor principal recibe una señal SIGHUP. Si se edita el archivo en una sistema activo, es necesario enviarle una señal al servidor (usando pg_ctl reload o kill -HUP) para hacer que vuelva a leer el archivo.

Nota: Para conectarse a una base de datos particular, un usuario no solo debe pasar las validaciones del pg_hba.conf, sino que también debe tener el privilegio CONNECT a la base de datos. Si desea restringir cuales usuarios pueden conectarse a cuales bases de datos, usualmente es más fácil controlarlo con la asignación/revocación del privilegio CONNECT que poner las reglas en el archivo pg_hba.conf.

Ejemplos de registros del archivo pg_hba.conf

1. Permite que cualquier usuario local se conecte a cualquier base de datos con cualquier nombre de usuario usando socket de dominio UNIX.

```
# TYPE  DATABASE    USER        CIDR-ADDRESS          METHOD
   local   all              all           trust
```

2. Igual que el anterior pero usando conexiones TCP/IP de loopback local

```
# TYPE  DATABASE    USER        CIDR-ADDRESS          METHOD
   host    all              all           127.0.0.1/32          trust
```

3. Lo mismo que el ejemplo 2, pero usando una columna con máscara de red.

```
# TYPE  DATABASE    USER        IP-ADDRESS    IP-MASK           METHOD
   host    all              all           127.0.0.1     255.255.255.255 trust
```

4. Permite que cualquier usuario desde cualquier equipo con dirección IP 192.168.93.x se

conecte a la base de datos “postgres” con el mismo nombre de usuario que *ident* reporta para la conexión (típicamente el nombre de usuario de Unix)

```
# TYPE  DATABASE    USER        CIDR-ADDRESS          METHOD
   host    postgres        all          192.168.93.0/24         ident
```

5. Permite que cualquier usuario desde el equipo 192.168.12.10 se conecte a la base de datos postgres siempre y cuando el password que este envía sea el correcto.

```
# TYPE  DATABASE    USER        CIDR-ADDRESS          METHOD
   host    postgres        all          192.168.12.10/32        md5
```

6. Si en el archivo no existen los registros de los ejemplos 2 al 5, las dos lineas que a continuación se describen rechazarán todas las conexiones desde 192.168.54.1, pero permite que peticiones de conexión Kerberos 5 desde cualquier otra parte de Internet. La máscara de ceros indica que ningún bit de las direcciones IP evaluadas son consideradas, por lo que coincide con cualquier equipo.

```
# TYPE  DATABASE    USER        CIDR-ADDRESS          METHOD
  host    all             all          192.168.54.1/32         reject
  host    all             all          0.0.0.0/0               krb5
```

7. Permite que se conecten usuarios que se trabajan en equipo que pertenecen a la red 192.168.x.x, siempre y cuando pasen la validación de ident. Si por ejemplo, si ident dice que el usuario es “juanito” y el solicita conectarse a PostgreSQL como “invitado”, la conexión se permite si existe una entrada en el mapa “omicron” que dice que “juanito” puede conectarse como “invitado”.

```
# TYPE  DATABASE    USER        CIDR-ADDRESS          METHOD
  host     all             all          192.168.0.0/16          ident map=omicron
```

8. Si estas fueran las únicas tres líneas para conexiones locales, permitirían a los usuarios locales conectarse solo a sus propias bases de datos (bases de datos con el mismo nombre que su usuario en PostgreSQL) excepto para administradores y miembros del grupo “soporte”, quienes pueden conectarse a todas las bases de datos. El archivo $PGDATA/admins contienen una lista con los nombres de los administradores. Se requiere del password en todos los casos.

```
# TYPE  DATABASE    USER        CIDR-ADDRESS          METHOD
  local   sameuser       all                               md5
  local   all             @admins                           md5
  local   all             +support                          md5
```

Creación de programas y problemas.

1. Cadenas de comandos SQL.

Una de las características de PostgreSQL y de algunas otras bases de datos es la posibilidad de escribir múltiples comandos en una sola cadena de texto. Esto es muy fácil de comprobar copiando la siguiente cadena al PSQL, debe notar que son dos consultas en la misma cadena, la intención es que la BD las ejecute a ambas, como si fuera una sola consulta. ¿Funciona?¿Cuál es el resultado?

```
select * from materias; select * from profesores; drop table CalendarioEscolar;
```

2. Creando las tablas de control.

Ahora construiremos las tablas que son necesarias para el trabajo en el sistema de menúes que va a controlar el acceso al sistema universitario. Use la misma base de datos que ha usado para la Universidad ACME.

```
create table usuarios(
   curp varchar primary key,
   nombre varchar,
   password varchar,
   login varchar
);

insert into usuarios values('sahe251287','Ervin Sanchez Hernandez', 'abc123','esanchez');
insert into usuarios values('aaol690820','Luis Antonio Alvarez Oval', '123098','lalvarez');
insert into usuarios values('caec850101','Christian Mauricio Castillo Estrada', 'cmce','ccastillo');
```

3. Preparando los programas para la autenticación de los usuarios.

Antes de efectuar el procedimiento de inyección es necesario construir los programas en Java que efectúan la autenticación de usuarios, los cuales tienen omisiones en su construcción y que en apariencia están bien construidos. Para ello se ocupan tres clases, Conexión, Asigna y Usuarios, para ejecutarlos su base de datos debe estar lista para recibir consultas, de preferencia utilice la que ha estado usando para el proyecto de Universidad ACME. Así que construya las clases necesarias en su proyecto de Netbeans y copie el código que se proporciona para cada clase. Verifique que sean correctos los paquetes en los que se ha construido las clases anteriores con las propias y modifique en caso de ser necesario.

Clase conexión: Clase que se ha usado en el laboratorio 4 y que ha modificado de acuerdo a las necesidades de su equipo, la cual permite a Java conectarse a la base de datos PostgreSQL.

Clase Asigna: Ejecuta la consulta a la BD del usuario que intenta iniciar el uso del sistema y despliega el mensaje “Usuario aceptado” si el usuario ha capturado su login y clave secreta correctamente aunque en lugar del mensaje debería estar la llamada a la clase que despliega el menú de las opciones del sistema de la universidad ACME. En caso de que este no haya capturado los datos correctamente se le despliega el mensaje “Usuario no aceptado”

Clase Usuarios: Genera la ventana donde el usuario login y clave secreta.

Clase Asigna – valida el login del usuario.

```
package oval;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import javax.swing.JOptionPane;

public class Asigna {
    public String login;
    public String password;
    private PreparedStatement registro;
    private ResultSet var;
    private Conexion pvedo = null;

    public void AsignaLogin(String login){
        this.login=login;

    }
    public void RecibePasswor1(String password){
        this.password=password;

    }

     public void Consulta(){

         pvedo= new Conexion();
        if(pvedo.b==true){
          try{
                registro =pvedo.con.prepareStatement("select * from usuarios where
login='"+this.login+"'and password='"+this.password+"';");
                // Despliegue aquí el contenido de la variable registro.
                var=registro.executeQuery();
                if(var.next()==true)
                {
                    JOptionPane.showMessageDialog(null,"Usuario
Aceptado","Bienvenido al sistema", JOptionPane.OK_OPTION);
                }else{
                    JOptionPane.showMessageDialog(null,"Usuario
Rechazado","Login o Password con errores, NO se permite el acceso al sistema",
JOptionPane.ERROR_MESSAGE);
                }
            }catch(Exception e){
                JOptionPane.showMessageDialog(null,e.getMessage(),"ERROR NO
SE PUEDE CONECTAR", JOptionPane.ERROR_MESSAGE);
                        }
        }
```

```
}
}
```

clase Usuarios, pantalla de acceso.

```
package oval;
import oval.Asigna;
import javax.swing.JOptionPane;

public class Usuarios extends javax.swing.JFrame {
    private Asigna bu= new Asigna();
    /** Creates new form Usuarios */
    public Usuarios() {
        initComponents();
    }

    // <editorfold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {
        jPanel1 = new javax.swing.JPanel();
        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        login = new javax.swing.JTextField();
        password1 = new javax.swing.JPasswordField();
        jButton1 = new javax.swing.JButton();
        jButton2 = new javax.swing.JButton();
        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_
CLOSE);
        jPanel1.setBorder(javax.swing.BorderFactory.createTitledBorder(""));
        jLabel1.setText("login");
        jLabel2.setText("Password");
        jButton1.setText("Aceptar");
        jButton1.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                jButton1MouseClicked(evt);
            }
        });
        jButton2.setText("salir");
        jButton2.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                jButton2MouseClicked(evt);
            }
        });
        javax.swing.GroupLayout jPanel1Layout = new
javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.
```

```
LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLay
out.Alignment.TRAILING, false)
                    .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
jPanel1Layout.createSequentialGroup()
                        .addGap(64, 64, 64)
                        .addComponent(jButton1)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlaceme
nt.RELATED, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addComponent(jButton2))
                    .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
jPanel1Layout.createSequentialGroup()
                        .addGroup(jPanel1Layout.createParallelGroup(javax.swing.Grou
pLayout.Alignment.LEADING)
                            .addGroup(jPanel1Layout.createSequentialGroup()
                                .addGap(53, 53, 53)
                                .addComponent(jLabel1,
javax.swing.GroupLayout.PREFERRED_SIZE, 42,
javax.swing.GroupLayout.PREFERRED_SIZE))
                            .addGroup(jPanel1Layout.createSequentialGroup()
                                .addGap(29, 29, 29)
                                .addComponent(jLabel2)))
                        .addGap(23, 23, 23)
                        .addGroup(jPanel1Layout.createParallelGroup(javax.swing.Grou
pLayout.Alignment.LEADING, false)
                            .addComponent(password1)
                            .addComponent(login,
javax.swing.GroupLayout.DEFAULT_SIZE, 170, Short.MAX_VALUE))))
                .addContainerGap(113, Short.MAX_VALUE))
        );
        jPanel1Layout.setVerticalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.
LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGap(37, 37, 37)
                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLay
out.Alignment.BASELINE)
                    .addComponent(jLabel1)
                    .addComponent(login,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(54, 54, 54)
                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLay
out.Alignment.BASELINE)
                    .addComponent(jLabel2)
                    .addComponent(password1,
```

```
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(98, 98, 98)
                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLay
out.Alignment.BASELINE)
                    .addComponent(jButton1)
                    .addComponent(jButton2))
                .addContainerGap(49, Short.MAX_VALUE))
        );
        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADIN
G)
            .addGroup(layout.createSequentialGroup()
                .addGap(22, 22, 22)
                .addComponent(jPanel1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(25, Short.MAX_VALUE))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADIN
G)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanel1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(17, Short.MAX_VALUE))
        );
        pack();
    }// </editorfold>
    private void jButton2MouseClicked(java.awt.event.MouseEvent evt) {
this.dispose();
    }
    private void jButton1MouseClicked(java.awt.event.MouseEvent evt) {
                bu.AsignaLogin(login.getText().toString());
             bu.RecibePasswor1(password1.getText().toString());
            bu.Consulta();


    }
```

```
    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new Usuarios().setVisible(true);
            }
        });
    }

    // Variables declaration  do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JTextField login;
    private javax.swing.JPasswordField password1;
    // End of variables declaration

}
```

4. Verificación de los programas.

Ahora probaremos que nuestros programas funcionan correctamente y finalmente ejecutaremos la inyección de SQL. Capture los valores para Login y Password en la pantalla de la clase Usuarios que se le indica en las secciones A, B, C y D del apartado 4.

A. Usuario que no existe.

Capture en la pantalla que generan el programa anterior los siguientes valores login: loval y en password: abc123. ¿Que mensaje se despliega? ¿Existe el usuario en la tabla? Debido a que el usuario loval no existe, es correcto que el sistema me niegue el acceso.

```
Login: loval
Password: abc123
```

B. Usuario que si existe:

Capture en la pantalla que generan el programa anterior los siguientes valores login: esanchez y en password: abc123. ¿Que mensaje se despliega? ¿Existe el usuario en la tabla? Debido a que el usuario esanchez existe en la tabla de control y a que el password que capturado es el correcto el programa me permite el acceso.

```
Login: esanchez
Password: abc123
```

Debido a que las pruebas a las que hemos sometido a los programas previos, podemos concluir que funcionan adecuadamente para nuestros cometido: evitar que usuarios que

no pertenecen a la Universidad ACME accedan a la información del sistema.

5. Inyección de SQL.

Ahora vamos a efectuar un pequeño cambio en la información que se captura en el campo Password, esta pequeña diferencia es lo que se llama Inyección de SQL y que es el tema central de este laboratorio.

C. Inyección de SQL – Engañando al SQL.

Capture en la pantalla que generan el programa anterior los siguientes valores login: lalvarez y en password: **' or '1'='1**. ¿Que mensaje se despliega? ¿Existe el usuario en la tabla? Obviamente el password está mal, ya que el password registrado es 123098. ¿Que pasó con el programa?

Login: lalvarez Password: **' or '1'='1**

Para saber lo que está sucediendo vaya a la clase Asigna y en el método Consulta() imprima el valor que asume registro al capturar los campos login y password. Escriba la cadena SQL que se arma en el método consulta de la clase Asigna. ¿Cuál es el problema? ¿Por que permite el acceso como si fuera un usuario válido?

D. Inyección de SQL – Eliminando una tabla.

Una vez que ya entendió la razón por la cual funciona la inyección de SQL, vamos a efectuar una inyección mucho mas maliciosa. Intentemos borrar una tabla desde la inyección de SQL. ¿Se puede? Capture los valores de la tabla de abajo en la pantalla de captura que ha usado para este ejercicio.

Login: loval Password: **' or '1'='1'; drop table CalendarioEscolar;'**

Después de haber aplicado esta segunda inyección, intente consultar en el PSQL el contenido de la tabla CalendarioEscolar (use el comando select * from CalendarioEscolar) ¿Existe? ¿En que momento se elimino? Escriba la cadena SQL que se arma en el método consulta de la clase Asigna.

E. Eliminando el problema.

En el artículo “SAFELI – Escáner de Inyección de SQL usando Ejecución Simbólica” [Fu, Qian] nos dan una solución para eliminar el problema de inyección de SQL. La función massage(). En cuya primera parte se debe hacer la revisión de los comandos SQL sospechosos, tales como OR, DROP, etc. Si cualquiera de ellos es encontrado, la función devuelve una excepción. La segunda parte tiene que ver con el tristemente celebre carácter de comilla simple. Note que en lenguaje SQL, la comilla simple puede ser usada como un carácter de datos usando secuencias de escape. La forma de escape de la comilla simple es “””. La función massage() sustituye cada comilla simple con “””, por ejemplo, el caracter de escape. Entonces la comilla simple no será tratada como un caracter de control. Finalmente, massage() intenta proveer protección adicional

restringiendo la salida a la longitud de 16 caracteres. Dese cuenta que “16” es simplemente un número constante mágico, su motivación es limitar la posibilidad de que los usuarios puedan hacer ataques.

Esta función debe ser usada de la siguiente manera: toda captura de datos del usuario debe de pasar por la función massage() antes de ser incrustada en una consulta SQL.

```
Public String massage(String strInput){
    //1. Busqueda de comandos SQL
    if(strInput.IndexOf("- -")!=-1
       || strInput.IndexOf("OR")!=-1
       || strInput.IndexOf("drop")!=-1)
            throw new Exception( "Posible ataque con inyección de SQL: " + strInput );
    //2. Dar masaje a los datos buscando la comilla simple
       String sOut = strInput.Replace("'","''");
       sOut = sOut.Substring(0,16);
    return sOut
}
```

Agregue la función a la clase asigna y ahora intente la inyección de SQL con las nuevas modificaciones. ¿Ha eliminado el problema?

Trabajo adicional.

1. Escriba la función masaje que evite que usuarios maliciosos puedan usar inyección de SQL. En los programas que hemos estado usando, aplique la función masaje a la captura de Login y Password y solo después arme la consulta de SQL que envía a validar a la base de datos. ¿Funciona? ¿Es útil la función para todos los casos de captura de información?¿Cómo debo de mejorar la función? Para esta pregunta espero que además de sus respuestas me entregue el código fuente con las mejoras que haya sugerido.

2. Explique y de un ejemplo como se puede ejecutar una inyección de SQL usando lenguaje HTML sobre una página Web. Monte el servidor Web sobre un equipo y el servidor de base de datos PostgreSQL en otro. Configure PostgreSQL para que solo acepte consultas desde el equipo donde está montado el servidor Web, además solo debe aceptar que se conecte el usuario *webadmin* y que el password lo envíe encriptado.

3. Explique y de un ejemplo como se puede ejecutar una inyección de SQL usando lenguaje PHP sobre una página Web.

4. Busque en Internet herramientas que prueben los programas de Java con la intención de detectar posibles vulnerabilidades de inyección de SQL. De ser posible verifique los programas que ha construido con esa herramienta.

5. Construya el sistema que resuelve el problema de la universidad ACME. Agregue un menú desde el cual se acceda a las pantallas. Asegúrese de que los usuarios solo

tengan acceso a las pantallas que modifican las tablas a las que tienen derecho. La conexión a la base de datos debe hacerse de acuerdo al usuario (Login) que está usando en el sistema. Modifique las tablas para que en todos los casos se almacene, sin que el usuario lo sepa, login, fecha y hora de cada cambio a los registros, esto por cuestión de seguridad. Antes de mostrar el menú se debe requerir el login y password para el usuario; esta pantalla debe estar blindada contra inyecciones de SQL. Describa como va a controlar el acceso para cada usuario a las ventanas, ya que es posible que se desarrollen nuevas opciones y además es posible que el personal rote entre departamentos. Así que la administración del sistema debe ser muy versátil. Aplique la política descrita por la teoría para evitar la inyección de SQL: aplique la función masaje a todos los campos en los que el usuario captura información.

Referencias

• PostgreSQL Introduction and Concepts by Bruce Momjiam
• PostgreSQL by Susan Douglas and Korry Douglas.
• Manual de postgreSQL: www.postgresql.org
• Fundamentos de Base de Datos. 5a. Edición de Abraham Silberschatz. Ed. McGraw-Hill
• Elmasri y Navathe: Fundamentos de Sistemas de Bases de Datos - 3a edición, 2002.
• Garcia-Molina, Ullman y Widom: “Database systems: the complete book”. Prentice-Hall.