# La seguridad en aplicaciones con acceso SQL

Parece que cuando uno lleva ya algún tiempo trabajando en seguridad informática, debería inmunizarse contra la pasividad de determinados fabricantes y clientes ante la existencia de gravísimos fallos de seguridad en sus programas. No es mi caso. Quienes llevamos tiempo en el sector tenemos todavía que luchar contra viejos clichés (algunos de ellos extendidos por los fabricantes cuyas herramientas recomendamos) que aún hoy persisten en muchos de nuestros clientes. Uno de los grandes caballos de batalla es la seguridad a nivel de aplicación, y la implicación de los responsables de seguridad en los proyectos software. A lo largo del artículo intentaré explicar a través de ejemplos por qué es importante la seguridad en aplicaciones con acceso SQL, y algunas recomendaciones útiles para "securizarlas".

Jorge Hurtado

Es curioso comprobar como con el paso del tiempo las vulnerabilidades en el software siguen aún apareciendo como si estuviésemos todavía en la edad media de la seguridad en Internet. La ingente información disponible para los programadores sobre la necesidad de la validación y control de los parámetros no parece suficiente, pues siguen cometiendo los mismos errores que antaño, codificando programas inseguros que posibilitan el acceso completo a sus sistemas de información. Tampoco parece intimidar a los fabricantes de software comercial el hecho de que miles de ojos escudriñen el funcionamiento de sus aplicaciones a la búsqueda del más mínimo fallo que aprovechar, ya que asisten impasibles al descubrimiento de más y más errores que hacen a nuestros sistemas vulnerables.

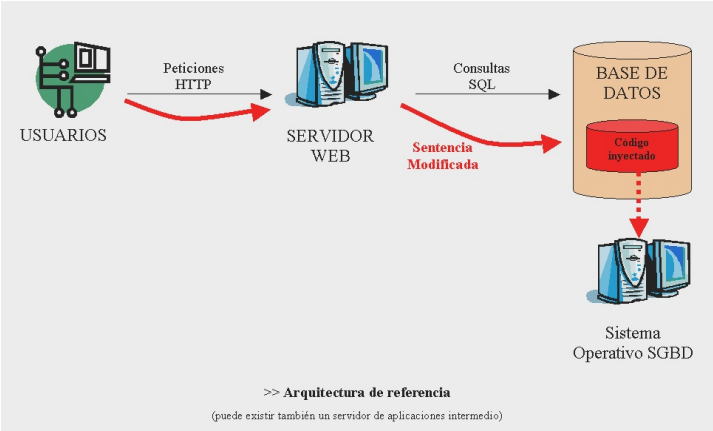
Es irónico comprobar como aún hoy en día grandes sitios web de todos los sectores aluden a la seguridad en sus servicios web y la explican con el ya mítico candado del navegador, hablando del número de bits y la robustez de sus claves, mientras un mínimo análisis de su aplicación sería capaz de sacar los colores al más indolente responsable de seguridad informática.

Tampoco es raro seguir encontrando grandilocuentes ofertas para realizar intrusiones informáticas, cuando lo que se está intentando vender es un informe generado con una conocida herramienta comercial, sin un mínimo análisis adicional de los resultados.

En absoluto esto significa que las herramientas comerciales no sean útiles, si no que abordan sólo una pequeña parte de la seguridad. ¿Acaso nos van a analizar la seguridad de nuestros aplicativos, la de nuestros procedimientos, incluso la de nuestra seguridad perimetral?

Es en esta falsa sensación de seguridad dónde se encuentran la mayoría de empresas de nuestro país: *"tengo un cortafuegos, me ha hecho una auditoría una firma de prestigio, e incluso dispongo de licencia de un conocido producto comercial para realizar auditorías internas. No puedo estar más seguro"*

Y es en esta situación cuando al realizar una prueba de hacking ético por parte de una empresa especializada como Germinus Solutions, se muestra más evidente el desespero de los directores de seguridad informática, que asisten impotentes a la violación de sus sistemas de información, protegidos por millonarias medidas de seguridad. Uno se da cuenta entonces en que sólo a través de esta política de hechos consumados *("Hay un problema porque esta empresa tiene control total sobre mi sistema")* es posible concienciar no sólo a los responsables de seguridad, si no también a los directores de sistemas, incluso a la Dirección de las compañías, de la importancia de la seguridad a nivel de aplicación, y de lo inútil que resultan las medidas de seguridad aisladas, por mucho nombre (y precio) que tengan.




>> **Arquitectura de referencia**

(puede existir también un servidor de aplicaciones intermedio)

Pecando de una cierta ingenuidad, tal vez el hecho de explicar la importancia de la seguridad a nivel de aplicación, esta vez con malos ejemplos de cómo *no* se deben hacer las cosas, pueda servir a los sufridos responsables de seguridad de las empresas a concienciar (se) al resto de departamentos de que su labor no sólo consiste en el diseño de medidas de seguridad, si no que debe existir una implicación total de los responsables de seguridad dentro del desarrollo de las aplicaciones.

Vamos además a analizar uno de los sistemas más fácilmente disponibles y quizá también más olvidados, los sistemas de bases de datos utilizados para proporcionar un contenido dinámico a las páginas web. En general está extendida la falsa creencia de que estos sistemas son difíciles de vulnerar a nivel de aplicación y que para ello se requiere un gran conocimiento de la estructura de la base de datos, siendo posible en el mejor de los casos actuar de forma destructiva. Trataremos de desmitificar esta aparente inocuidad a través de un ejemplo sencillo.

Supongamos en primer lugar un programa web con el siguiente pseudocódigo:

```
<Recupera variables de usuario>
<Construye Sentencia SQL:
s= 'select * from noticias where idnot='
+ Parametro(idNot)
>
<Recupera Resultados>
<Muestra resultados>
```

Serían, por ejemplo, las típicas notas de prensa dinámicas, y el código anterior el que aparece al pulsar sobre uno de los resúmenes para mostrar la noticia completa.

En principio no parece que el programa sea muy problemático. Ningún analizador de vulnerabilidades de los más conocidos va a detectar ningún problema, ya que tanto el servidor de web, como los elementos intermedios (cortafuegos, routers) se hayan perfectamente configurados y actualizados. Es muy posible que este cgi pase a producción, al comprobar desde el departamento de desarrollo que cumple con la funcionalidad exigida y desde el departamento de seguridad que al correr una exploración con un reconocido producto no ha disparado ninguna alarma.

Sin embargo la realidad es muy distinta, ya que dicho programa no comprueba que el parámetro recibido del navegador del usuario sea un parámetro correcto y lo introduce sin control en un intérprete SQL.

Ahora pongámonos el gorro (hay veces que es conveniente) de la persona que intenta aprovechar este tipo de fallos. Lo primero que haría es introducir una comilla simple como Parámetro "idNot".

```
ODBC Error XXXX. SQL Server Driver:
la cadena entrecomillada no está
terminada correctamente
```

Dos errores en uno ya que hemos

comprobado que por una parte no existe un control a nivel de aplicación y por otro que no existen programadas excepciones genéricas en caso de fallo, para evitar proporcionar tanta información sobre el sistema, por lo que también conocemos el gestor de base de datos utilizado.
Ahora que sabemos que tenemos acceso al intérprete SQL, vamos a pensar en qué daño o qué provecho podríamos sacar de esta situación. Supongamos ahora que "inyectamos" idNot=1 or 1 in (drop table noticias). La sentencia ejecutada sería:

```
select * from noticias where idnot=1 or
1 in (drop table noticias)
```

La sentencia anterior borraría la tabla noticias. Si ahora pensamos que en lugar de una tabla de noticias se tratase de una de pedidos o de clientes, las consecuencias serían mucho más graves. Otra posibilidad sería dar de alta noticias falsas, que podrían tener intenciones perversas, por ejemplo un comunicado del consejo de administración, o un *profit warning*.

Pero es verdad que mediante un *backup* reciente, se podría volver a prestar servicio en cuestión de minutos u horas. De alguna manera se trata de un ataque poco sofisticado. ¿Podríamos hacer algo más?

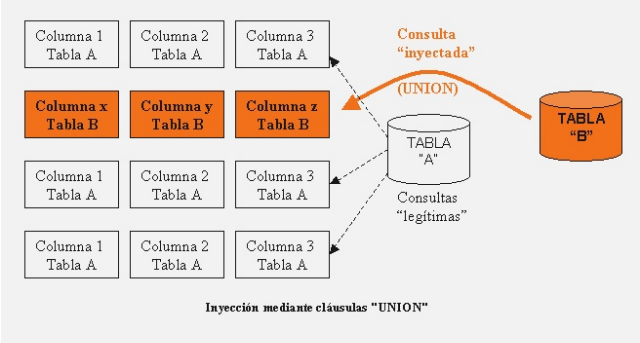
Probablemente podamos, aunque nos va a exigir un mínimo conocimiento del gestor de base de datos y en particular de los procedimientos almacenados que nos proporciona. Sigamos con nuestro ejemplo, en el que el intérprete de la base de datos es SQL Server. Este intérprete cuenta con multitud (infinidad) de procedimientos almacenados que pueden, en función de la configuración, ser accesibles o no. El procedimiento almacenado master..xp_cmdshell ('comando') permite ejecutar sentencias del sistema operativo a través de una consulta SQL. De hecho si ejecutásemos el siguiente comando:

```
select * from noticias where idnot=1 or
1 in (EXEC master..xp_cmdshell 'ftp -
I ...')
```

Con los parámetros del ftp adecuados, podríamos recuperar un troyano de internet. Para los escépticos, si el sistema estuviese oculto tras una pantalla NAT, podríamos construir a través de sucesivos types, un cgi comando (.bat) que llamase al intérprete del sistema operativo y nos diese la salida del comando por pantalla, incluso a través de la adecuada utilización del comando NET, podríamos reconfigurar gran parte de los parámetros del servidor para que fuese más fácil nuestra entrada.

Por supuesto lo dicho es aplicable para otros sistemas operativos e intérpretes de bases de datos, un error a nivel de aplicación, que ha cometido por descuido o ignorancia un desarrollador, compromete gravemente la seguridad de los sistemas de información, afectando a los responsables de explotación, al de seguridad, e incluso pudiendo llegar a tener graves repercusiones sobre la imagen de la empresa.

¿Esto es todo? En realidad no. Supongamos que o bien no damos con el intérprete que conocemos, o bien su configuración es la adecuada y no nos permite la ejecución de comandos a a través de procedimientos almacenados. ¿He de quedarme ahí? ¿Se me ha terminado la suerte? Puede que no. Ahora vamos a utilizar las uniones de SQL para intentar recuperar información de la base de datos a la cual en principio



Inyección mediante cláusulas "UNION"

Mecanismos de Inyección SQL

no tendríamos acceso, incluso información que a pesar de residir en el mismo motor de base de datos, no se ha publicado en Internet porque se trata de información interna, o que está destinada a una extranet para clientes o partners.

Se trata de conseguir, a través de una unión con otra consulta, los datos que en realidad nos interesan. Para ello la consulta "inyectada" debe tener el mismo número de parámetros que la original, si bien esta dificultad puede salvarse mediante el conocido método de "prueba y error". Supongamos que conozco la tabla de los datos que quiero sacar (luego veremos también cómo obtener la estructura de la base de datos con el mismo sistema).

```
IdNot= 76436487 union select nombre,
tarjeta, caducidad, caducidad,
caducidad from pedidos where
idcliente=3456
```

La sentencia ejecutada es:

```
select * from noticias where
idnot=76436487 union select nombre,
tarjeta, caducidad, caducidad,
caducidad from pedidos where
idcliente=3456
```

Por supuesto al no haber una noticia con ese índice, la consulta tiene los datos de la tabla pedidos, y en la página de detalle de la noticia en realidad veríamos los datos de la tarjeta de crédito del cliente 3456.

Lo que no sería tan fácil es adivinar la estructura de la base de datos ¿Cómo la recupero? Pues a través de consultas a las tablas del sistema, que sí son conocidas aunque distintas para cada motor de base de datos y que nos proporcionarán detallada información sobre la información que puedo obtener, por ejemplo:

```
select * from noticias where
idnot=76436487 union select
tablename, columnname, columnname,
columnname, columnname from
sysmaster where tableindex=1
```

Con lo que sucesivamente iría obteniendo las columnas y las tablas de la base de datos.
Llegados a este punto, debido a un descuido en la programación, hemos borrado tablas, insertado información en la base de datos, accedido al sistema operativo y recuperado valiosa información de la base de datos.

¿Qué han hecho nuestros sofisticados sistemas de seguridad en todo este proceso?

- El Cortafuegos ha analizado el tráfico y al comprobar que se trata de una conexión http correcta la ha dejado pasar.
- El Detector de Intrusos no lo reconoce como un intento de ataque, puesto que a sus ojos se trata de una petición "normal" a un CGI, y por supuesto no coincide con su base de datos de ataques.
- Si existe detector de integridad o de intrusos en el servidor final, no realizará ninguna alerta mientras no modifique ningún parámetro del sistema, pero incluso un ataque elaborado podría desactivar los demonios de protección una vez se tuviese acceso al sistema operativo
- Los administradores de la empresa tampoco pueden hacer nada, ya que para ellos el flujo de sentencias de ejecución SQL contra la base de datos es normal (por supuesto a no ser que se actúe de una forma destructiva)

En realidad todas las piezas han hecho lo que deben hacer y desgraciadamente no les podemos pedir más, ya que por muy grandilocuente que resulte la publicidad de los distintos fabricantes, un error a nivel de aplicación pasa desapercibido por ellos.

Es pues necesaria una estrecha coordinación entre los responsables de seguridad informática de las grandes instituciones y los departamentos de Ingeniería y desarrollo. Esta coordinación debería plasmarse en:

- Guia de Programación segura *en cada uno de los entornos* en los que se trabaje.
- Auditorías de Código Internas y Externas (de Caja Blanca).
- Diseño de un marco de seguridad para las aplicaciones institucionales, que cubra aspectos como la autenticación, no repudio, confidencialidad de la información, control de acceso, registro de sucesos y auditoría.
- Formación a los desarrolladores sobre aspectos de seguridad.
- Participación directa en aquellos proyectos en los que la seguridad sea crítica, colaborando en todo el ciclo de vida del proyecto software.

Por último, esta necesaria relación entre departamentos tradicionalmente disjuntos (cuando no enfrentados) sienta las bases para una mayor seguridad en las aplicaciones (externas o internas) y, si bien no garantizan la invulnerabilidad de las aplicaciones desarrolladas, al menos proporcionan un marco de seguridad homogéneo en las aplicaciones corporativas, pudiendo evitar (o al menos detectar) la mayor parte de los ataques que hemos descrito en este artículo.

Jorge Hurtado
Director de Desarrollo de Negocio
GERMINUS SOLUTIONS
jhurtado@germinus.com