SFX-SQLi

SELECT FOR XML SQL INJECTION

Extracción rápida de información utilizando inyección SQL con instrucciones XML

Daniel Kachakil (dani@kachakil.com)

5 de febrero de 2009

Contenido

Resumen

A medida que los gestores de bases de datos van evolucionando con nuevas versiones dotadas de mayor funcionalidad, van apareciendo nuevos métodos de ataque que aprovechan estas capacidades. Este documento detalla un método para extraer toda la información de una base de datos de **Microsoft SQL Server** utilizando una técnica muy eficiente basada en el uso de la cláusula **FOR XML**, de la cual se presenta una herramienta que la implementa.

Esta técnica no descubre un nuevo tipo de vulnerabilidad, sino que se basa en el mejor aprovechamiento de una técnica conocida y utilizada desde hace ya 10 años: la inyección de código SQL (***SQL injection***). A pesar de su antigüedad y de la publicación constante de numerosas técnicas de ataque y herramientas automatizadas basadas en esta vulnerabilidad, a día de hoy sigue siendo una de las más extendidas en las aplicaciones web, comprometiendo la confidencialidad, la integridad y la disponibilidad de gran cantidad de datos.

Asumiendo un escenario en el que sea posible aplicar técnicas de inyección SQL, trataremos de obtener una inyección con una instrucción de unión válida que sea capaz de anexar un registro a los resultados de alguna consulta vulnerable de una aplicación web. El principal requisito es que podamos leer al menos uno de sus campos, preferiblemente de tipo texto (a través de HTML, como respuesta de un servicio web, exportado a un fichero, etc).

Si conseguimos mostrar aunque sea un solo carácter a nuestro antojo, en el caso general será posible sustituir dicho carácter por el contenido de una tabla completa, siempre que podamos volcarla en una única cadena de texto. Para ello utilizaremos la cláusula **FOR XML** de la instrucción **SELECT** incorporada en **Microsoft SQL Server** desde su versión 2000.

Introducción: la inyección de código SQL

El primer documento con referencias a la inyección de código SQL aparece a finales del año 1998 [1], introduciendo el problema con algunos ejemplos básicos, pero mencionando también el mayor impacto potencial si se saben aprovechar las potentes características de algunos procedimientos almacenados (por ejemplo, el envío de los datos por correo electrónico). En ese mismo documento se describe la forma de protegerse ante este tipo de ataques y dichos mecanismos de protección siguen siendo perfectamente válidos y no han variado en absoluto a día de hoy.

Con el paso de los años han ido surgiendo nuevas técnicas que han sabido exprimir al máximo este tipo vulnerabilidades, aprovechando todo tipo de facilidades que se le ofrecen al desarrollador y que a su vez suelen ser muy útiles desde el punto de vista del atacante. Desde una simple sintaxis que permite comentar parte de la consulta hasta la existencia de complejos procedimientos almacenados predefinidos capaces de enviar correos electrónicos, de volcar información a disco o de ejecutar comandos del propio sistema operativo… todo puede ser de gran utilidad para un auditor y a su vez para un posible atacante.

Este tipo de vulnerabilidad genérica no solo está presente en Microsoft SQL Server, sino que afecta a todo tipo de gestores de bases de datos relacionales (Oracle, MySQL, etc), ya que no es un fallo propio del gestor de bases de datos ni de la tecnología de desarrollo utilizada, sino de la forma inadecuada o inexistente de filtrar las entradas del usuario a través de la aplicación web por parte del programador, así como de la política de permisos implementada a nivel de los servidores, que podría agravar su impacto.

Las primeras técnicas que aparecieron sobre el volcado masivo de información, de envío de correos electrónicos o de ejecución de comandos son difícilmente aprovechables si el servidor está correctamente configurado a nivel de permisos. Sin embargo, a día de hoy existen multitud de herramientas que consiguen extraer toda la información de una base de datos de forma completamente automatizada, aprovechando diferentes variaciones sobre la misma técnica de inyección de código SQL, sin requerir ningún tipo de privilegios especiales.

Por ejemplo, si la aplicación web muestra los mensajes de error del gestor de bases de datos, puede resultar factible la extracción de información forzando la aparición de errores de conversión de tipos intencionados que pueden revelar todo tipo de información acerca de la estructura de la base de datos, así como de los propios datos que contiene [2][3].

Para los casos en los que la aplicación no muestre ningún tipo de mensaje de error descriptivo, también existen otras técnicas de inferencia de información a ciegas (*Blind SQL injection*) [4][5][6][7], basadas en la variación de la respuesta o del comportamiento de la aplicación web (variaciones en el código HTML devuelto, redirecciones a otras páginas, retardos, etc).

El principal inconveniente de la aplicación de este tipo de técnicas es la gran cantidad de peticiones y de tiempo que requieren para su ejecución, dejando además una traza considerable en los ficheros de registro de accesos del servidor, pudiendo incluso llegar a alertar a los administradores del servidor o del sitio web si comprueban que de repente se ha empezado a producir un notable aumento en los accesos, una degradación del rendimiento o un número de errores elevado por causas en principio desconocidas.

Las técnicas que se describen en el presente documento aprovechan una de las capacidades que tiene Microsoft SQL Server desde la versión 2000 (8.0), aunque ha sido mejorada en su versión 2005 y mantenida en la 2008, permitiendo una explotación del ataque de forma más sencilla y potente. Se trata de la cláusula FOR XML que se incluye dentro de la sintaxis de la cláusula SELECT.

Fundamentos: La cláusula FOR XML de la instrucción SELECT

La versión 2000 de Microsoft SQL Server introduce una nueva cláusula dentro de la sintaxis de la instrucción SELECT para especificar que el resultado de la consulta deberá devolverse en formato XML. Se trata de la cláusula FOR XML, que puede aparecer al final de la instrucción junto con una serie de argumentos opcionales.

Veamos un ejemplo de los resultados que devuelve la ejecución de las siguientes consultas en un SQL Server 2005 sobre una hipotética base de datos de prueba que contenga una tabla con algunos sencillos datos:

- `SELECT * FROM Numeros`

Numero	Nombre
1	Uno
2	Dos
3	Tres
4	Cuatro

- `SELECT * FROM Numeros FOR XML RAW`

XML_F52E2B61-18A1-11d1-B105-00805F49916B
<row Numero="1" Nombre="Uno" /><row Numero="2" Nombre="Dos" /><row Numero="3" Nombre="Tres" /><row Numero="4" Nombre="Cuatro" />

Se puede observar que la segunda consulta devuelve en esencia los mismos datos que la primera, pero con la diferencia de que la primera devuelve un conjunto de resultados, mientras que la segunda únicamente devuelve un único valor que contiene la misma información pero en formato XML. Ésta última incluye además los nombres de las columnas que aparecen como atributos, lo cual nos permitirá extraer directamente la estructura de la tabla (aunque por ahora podríamos deducir el tipo de datos de cada columna por los datos que contiene, más adelante veremos cómo extraer esta información de otras maneras).

Supongamos la existencia de una aplicación que muestre la representación en texto de un dígito usando la siguiente consulta vulnerable a inyección SQL:

- `"SELECT Nombre FROM Numeros WHERE Numero=" + Params("num")`

Lo que se describe a continuación es básicamente la explicación a qué sucedería si inyectáramos en nuestra aplicación de ejemplo una cadena como la siguiente:

- `-1 UNION SELECT (SELECT * FROM Tabla FOR XML RAW)`

Microsoft SQL Server 2005 y 2008

Si nos remitimos a la documentación del producto, encontraremos una serie de argumentos opcionales que se le pueden aplicar a la cláusula FOR. Esta es la sintaxis que aparece en la documentación oficial [8]:

```
[ FOR { BROWSE | <XML> } ]
<XML> ::=
XML
{
    { RAW [ ( 'ElementName' ) ] | AUTO }
    [
        <CommonDirectives>
        [ , { XMLDATA | XMLSCHEMA [ ( 'TargetNameSpaceURI' ) ] } ]
        [ , ELEMENTS [ XSINIL | ABSENT ]
    ]
  | EXPLICIT
    [
        <CommonDirectives>
        [ , XMLDATA ]
    ]
  | PATH [ ( 'ElementName' ) ]
    [
        <CommonDirectives>
        [ , ELEMENTS [ XSINIL | ABSENT ] ]
    ]
}

<CommonDirectives> ::=
[ , BINARY BASE64 ]
[ , TYPE ]
[ , ROOT [ ( 'RootName' ) ] ]
```

En primer lugar nos encontramos con la opción BROWSE, que nada tiene que ver con lo que buscamos y que, por tanto, no nos interesa en absoluto. Nos interesa la otra opción posible (es decir, XML), que a su vez admite un conjunto de argumentos de los cuales es obligatorio especificar únicamente uno de ellos. Se trata de RAW, AUTO, EXPLICIT y PATH. Veamos la utilidad que tienen para nuestro objetivo concreto:

- **RAW:** Muestra la información convirtiendo directamente cada fila del conjunto de resultados en un elemento de XML de la forma *<row />*. Opcionalmente se puede especificar otro nombre de elemento, que podría ahorrar algunos bytes en la respuesta si se utiliza un solo carácter.
- **AUTO:** Muestra la información de forma jerárquica para cada una de las tablas que aparezcan en la consulta. En nuestro caso no vamos a usar varias tablas en la misma consulta, por lo que esta opción no nos resultará de especial utilidad.
- **EXPLICIT:** Permite definir la forma en la que se representará el árbol XML, con una serie de condiciones muy particulares que tampoco aportan nada a la hora de conseguir el objetivo buscado.
- **PATH:** Permite definir la forma en la que se representará el árbol XML, con la idea de simplificar el modo EXPLICIT. Por defecto genera un elemento para cada fila y un elemento para cada columna, por lo que requeriría más espacio para representar la misma información comparando con el modo RAW.

De momento vemos que la opción RAW es la única que nos interesa, ya que el resto de opciones están diseñadas para tener mayor control sobre el formato en el que se va a generar la salida del árbol XML, pero en nuestro caso la primera opción es la más cómoda, teniendo en cuenta que a priori no conocemos la estructura de las tablas.

La opción RAW puede ir modificada por cualquiera de las directivas comunes que aparecen al final de la sintaxis, así como por otras particulares:

- **BINARY BASE64:** Especifica que la consulta devolverá los datos binarios (de tipo BINARY) codificados en formato BASE64. Al no ser la opción por defecto en el modo RAW, la consulta nos fallará en caso de encontrarnos con cualquier columna de tipo BINARY (que contenga imágenes, por ejemplo) si no especificamos explícitamente esta opción, por lo que a partir de ahora vamos a considerarla como opción imprescindible por este motivo.
- **TYPE:** Especifica que la consulta devuelve un resultado de tipo "xml" de SQL Server en lugar de devolverlo como texto. Como el objetivo es convertir la tabla en un texto, en principio no nos aporta nada el tipo de datos "xml".
- **ROOT:** Especifica si se va a añadir un elemento raíz al conjunto de datos devuelto. Por defecto generaría el elemento llamado *<root>*, aunque opcionalmente se le puede especificar un nombre diferente. Esta opción puede resultar muy útil a la hora de localizar dónde empiezan y dónde acaban los resultados de la consulta, lo cual no tiene mayor importancia si se inspecciona a mano, pero podría facilitar el trabajo de búsqueda a una herramienta que automatice el proceso.
- **XMLDATA:** Especifica que el esquema XDR debe aparecer en línea al principio de los resultados. Podría resultar de utilidad para determinar el tipo de datos, pero no muestra el tipo nativo de SQL Server, sino del esquema XML. En todo caso, esta opción falla si nos encontramos con una columna de tipo binario, por lo que su uso no es recomendable en general.
- **XMLSCHEMA:** Similar a la opción anterior, especifica que el esquema XSD debe aparecer al principio de los resultados. A diferencia del caso anterior, esta opción devuelve la estructura completa de los datos devueltos incluyendo el tipo en formato nativo, así como las restricciones de longitud máxima, de valor no nulo y de otras opciones. También es compatible con la directiva ROOT y con la existencia de datos en formato binario. Sin duda, se trata de una opción muy interesante, ya que devuelve toda la estructura de la tabla, detallando tipos de datos, nombres de columna y otras restricciones.
- **ELEMENTS:** Especifica que las columnas se devuelven como elementos XML (cuando por defecto se devuelven como atributos). En principio no tiene mucha utilidad (ya que el resultado requiere de más caracteres), siempre que no se tenga algún interés especial en determinar la existencia de los valores nulos, en cuyo caso deberíamos especificar también el argumento XSINIL (en caso contrario, simplemente se omitirían las etiquetas XML correspondientes a dichos campos).

La versión 2008 no introduce ningún cambio o novedad sobre la versión 2005 en cuanto a la cláusula FOR XML, más bien al contrario, ya que ahora se marca como obsoleta y deja de recomendarse su uso, a favor de los esquemas XSD. De todos modos, esto no supone ningún problema a día de hoy, ya que aún tendrán que pasar muchos años antes de que se migren todas las aplicaciones existentes a la futura versión posterior a la 2008, si es que finalmente decide eliminarse dicha funcionalidad en la siguiente versión (planificada para 2010), tal y como indica la documentación del producto.

Procedimiento general de extracción usando la cláusula FOR XML

Para lograr la extracción completa de una base de datos a través de un parámetro vulnerable a inyección SQL, en general se puede utilizar la técnica que se describe a continuación:

1. Ajuste de la inyección para canalizar la respuesta
2. Obtención del nombre de las tablas desde el catálogo
3. Obtención de la estructura y del contenido de cada tabla

Ajuste de la inyección para canalizar la respuesta

El primer paso consiste en obtener una inyección que permita manipular los resultados de alguna consulta para poder alojar en ella el contenido deseado. Para ello seguramente tengamos que utilizar una consulta existente y empezaremos por determinar su estructura interna (cantidad de parámetros y tipos de datos). Existen diversas técnicas documentadas para conseguir este fin, en función de si la aplicación muestra los errores, de si existe algún tipo de filtrado, de si podemos ejecutar ciertos comandos, etc.

Habiendo determinado esta estructura, en el caso general podremos anexar más registros usando una consulta de unión con el mismo número y tipo de parámetros. Si disponemos de los permisos adecuados, también podríamos actualizar o insertar algún registro de una tabla existente o crear una, volcar los datos a un fichero en disco, etc.

Lo que se describe en este punto no nada nuevo. Simplemente se trata un pequeño resumen de algunas técnicas conocidas usadas como requisito previo al funcionamiento del procedimiento de extracción descrito en este documento, de ahí su relevancia.

Determinación del número de columnas o parámetros de la consulta

Si la aplicación muestra el detalle de los errores que se producen en la base de datos, entonces nos encontramos ante el caso más sencillo, ya que podemos determinar con facilidad la estructura de una consulta e incluso los nombres de las columnas y tablas involucradas en la misma usando la técnica de HAVING 1=1 y GROUP BY [3][9].

- `HAVING 1=1` → Produce un error cuyo texto revela el nombre de la primera columna
- `GROUP BY columna1 HAVING 1=1` → Error con el nombre de la segunda columna
- `GROUP BY columna1, columna2 HAVING 1=1` → Error con el nombre de la tercera columna
- `...`
- `GROUP BY columna1, columna2, columna3, … , columnaN HAVING 1=1` → Sin errores

En caso de que la aplicación no muestre información sobre los errores, deberemos aplicar técnicas de inyección de SQL a ciegas (*Blind SQL injection*), entre las que por ejemplo podemos utilizar la cláusula ORDER BY seguida del número de columnas:

- `ORDER BY 1` → Sin errores
- `ORDER BY 2` → Sin errores
- `...`
- `ORDER BY N` → Sin errores
- `ORDER BY N+1` → Falla

También es válido el uso de la instrucción UNION seguida de tantos valores (normalmente nulos) como columnas existan [5][9]. Para esta técnica, si añadimos una condición falsa (ej: WHERE 0=1) a la consulta de unión, normalmente se facilitará el proceso ya que la fila de nulos inyectada no interferirá en los resultados.

- `UNION SELECT null WHERE 0=1` → Falla
- `UNION SELECT null, null WHERE 0=1` → Falla
- `UNION SELECT null, null, null WHERE 0=1` → Falla
- `...`
- `UNION SELECT null, null, null, null, … , null WHERE 0=1` → Sin errores

Determinación del tipo de las columnas obtenidas

Una vez obtenido el número de columnas que devuelve la consulta elegida, intentaremos determinar el tipo de datos de cada una de ellas, aunque para nuestro objetivo no siempre es imprescindible que este paso se complete con total precisión ni tampoco necesitaremos determinar el tipo de todas las columnas en la mayoría de los casos. Bastará con lograr inyectar un carácter en una columna de texto, siempre que éste sea procesado en la base de datos y devuelto por el servidor.

De nuevo, el caso más simple se dará cuando la aplicación muestre los errores, ya que podemos usar funciones como CAST() y CONVERT() para forzar una conversión de tipos y, en caso de fallo, el propio mensaje de error nos indicará el tipo de la columna (o la ausencia de dicho mensaje nos indicará que el tipo con el que hemos probado es correcto). Estas funciones se pueden usar dentro de una condición WHERE o en otra subconsulta con UNION.

En caso de que la aplicación no muestre errores, se usará inicialmente una consulta de unión con tantos valores nulos como columnas hayamos determinado. Posteriormente iremos probando a sustituir los nulos por diferentes tipos de datos (normalmente es suficiente con enteros y cadenas de texto). También podemos añadir una condición falsa antes y/o después de la consulta de unión, aunque de esta forma no siempre se obtendrán los resultados correctos, por lo menos es algo que nos podrá ayudar en el proceso.

En cualquier caso, no solo tendremos que ajustar los valores para que la consulta sea válida y no produzca ningún error en el gestor de base de datos, sino que la propia aplicación que utiliza esos datos tampoco podrá fallar en su ejecución (principalmente debido a encontrarse con valores nulos y otro tipo de valores inesperados), por lo que siempre será recomendable localizar una consulta lo más sencilla posible con el fin de evitar problemas ajenos a los fundamentos de la técnica de extracción aquí descrita.

Teniendo siempre presente nuestro objetivo, si en cualquier momento conseguimos inyectar y visualizar cualquier carácter o cadena de texto mientras estemos siguiendo este procedimiento, entonces podemos detenernos y pasar al siguiente punto.

Obtención de una consulta de unión adecuada

Habiendo obtenido una consulta de unión con el número y el tipo adecuado de valores en sus columnas y que no provoque ningún fallo en la base de datos, en ocasiones deberemos ajustar ciertos valores de forma manual para que la consulta no solo no provoque ningún fallo en la aplicación, sino que también consiga mostrar algún resultado de nuestra consulta.

Por ejemplo, si la aplicación solo muestra el primer resultado de la consulta, podemos conseguir que la consulta original de la aplicación no devuelva ningún registro usando una condición falsa antes de la unión (ej: WHERE 0=1). Si la aplicación falla o no devuelve resultados porque hemos introducido algún parámetro fuera del rango esperado, probaremos con otros más lógicos dentro del contexto de la aplicación, etc.

Teniendo ese punto resuelto, solo nos queda sustituir alguno de los valores de tipo texto por una subconsulta creada a nuestro antojo y comprobar que devuelva el resultado esperado. Siempre que sea posible, intentaremos encontrar un campo cuyo contenido no sea modificado por la aplicación (por ejemplo, codificándolo en HTML o limitando el número de caracteres), aunque incluso en la situación de no haber encontrado el caso óptimo, normalmente también podremos evitar que este tipo de restricciones afecten al proceso.

Obtención del contenido de una tabla

Para obtener el contenido completo de cualquier tabla, en general la estructura de la consulta a inyectar deberá tener este aspecto (siendo v_1-v_N los valores adecuados para que la aplicación devuelva resultados):

- `1 AND 1=0 UNION SELECT` v_1`,` v_2`, … , (SELECT * FROM Tabla FOR XML RAW, BINARY BASE64), … ,` v_N

Dependiendo del caso concreto, es posible que también se requiera alguna comilla para cerrar el primer valor si es de tipo texto, de un guión doble para ignorar el final de la consulta original de la aplicación, de reemplazar los espacios en blanco por un comentario vacío (/**/), de incluir el nombre del esquema antes del nombre de la tabla, etc.

El contenido obtenido será una larga cadena de texto que contendrá tantos elementos XML como filas devuelva la consulta inyectada. Puesto que un documento XML bien formado debe contener un único elemento raíz, deberemos añadirlo de forma manual siempre que no hayamos especificado la opción ROOT en la inyección.

Una de las formas más rápidas y cómodas de visualizar y trabajar con la información extraída en este formato es usando la clase DataSet de Microsoft .NET Framework [10], ya que permite importar directamente el XML y vincularlo a cualquier control como origen de datos, además de facilitar operaciones como su ordenación o filtrado. Al ser ambos productos del mismo fabricante y estar pensados para facilitar el trabajo a los programadores, podemos asumir que no deberíamos tener ningún problema de incompatibilidad.

Obtención de la estructura de una tabla

Para obtener la estructura de una tabla podemos usar la opción XMLSCHEMA de la propia cláusula FOR XML, podemos consultar el catálogo del sistema (ej: SYSOBJECTS) o podemos inferirla directamente a partir de los datos extraídos (que incluyen los nombres de las columnas). Normalmente será suficiente con aplicar el último método, siempre que no se desee hacer un muestreo previo para determinar las tablas y columnas que interese volcar.

Obtención del nombre de las tablas de la base de datos

Siempre que sea posible, intentaremos determinar los nombres de las tablas desde el catálogo de la propia base de datos usando vistas predefinidas del sistema como SYSOBJECTS o INFORMATION_SCHEMA.TABLES, volcando su contenido íntegro con la misma técnica que se describe en este documento. Una vez tengamos los nombres de tablas y esquemas, no hay más que volver a aplicar el mismo método para cada una de las tablas que queramos volcar.

Aquí se muestra una posible consulta que devolvería información suficiente acerca de las tablas existentes en la base de datos:

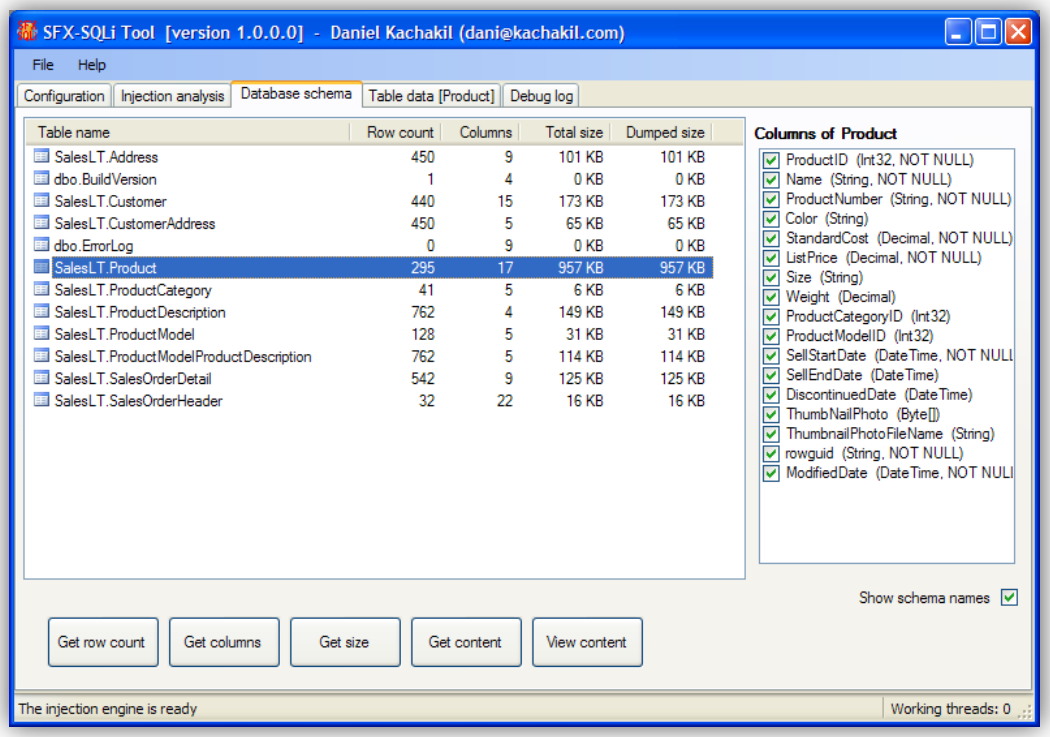
- `SELECT TABLE_SCHEMA, TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE Table_Type='BASE TABLE'`

Nótese que el método únicamente requiere de una sola petición al servidor por cada una de las tablas que queramos volcar (en el caso ideal).

Automatización: La herramienta SFX-SQLi

Como ya se ha comentado anteriormente en este documento, esta técnica puede automatizarse con relativa facilidad, ya que su aplicación en general no es un procedimiento especialmente complejo. Para demostrar el funcionamiento y el potencial de esta técnica, se ha desarrollado una herramienta que implementa todo lo que se ha ido describiendo en este documento. La herramienta SFX-SQLi permite obtener toda la estructura y el contenido de una base de datos de una web vulnerable y de forma casi totalmente automatizada (previo ajuste de la inyección), permitiendo observar y analizar lo que está sucediendo en cada momento.

Es posible que nos encontremos con alguna tabla con muchos registros o con datos binarios que ocupen mucho espacio y que el servidor de bases de datos no sea capaz de procesar en el tiempo máximo asignado a la ejecución de una consulta (o que la respuesta del servidor web se demore más de lo permitido, provocando que el método falle por causas ajenas al procedimiento). Por ello, entre otras características, la herramienta SFX-SQLi se encarga de optimizar el proceso, reduciendo el tamaño del XML devuelto (asignando alias a las columnas) y dividiendo la consulta por subconjuntos de filas (usando la función ROW_NUMBER) y por bloques de texto (usando la función SUBSTRING) para minimizar la carga del servidor, aunque sea a costa de incrementar el número de peticiones.



En su estado inicial, la herramienta SFX-SQLi carece de algunas características que incluso podrían considerarse como esenciales, pero hay que tener en cuenta ésta no ha sido diseñada para servir como herramienta de auditoría, sino como prueba de concepto.

Aplicación de la técnica en otros entornos

Aunque todo lo que se ha descrito hasta el momento está enfocado hacia aplicaciones web basadas en Microsoft SQL Server 2005/2008, algunas ideas que se presentan en este documento también son aplicables en otros entornos.

SQL Server 2000

La sintaxis admitida para esta versión de SQL Server es bastante más simple [11], debido probablemente a que se trata de la primera versión que introduce esta cláusula y a que la adopción del estándar XML tampoco estaba tan extendida por aquél entonces:

```
[ FOR XML { RAW | AUTO | EXPLICIT }
            [ , XMLDATA ]
            [ , ELEMENTS ]
            [ , BINARY BASE64 ]
]
```

En el caso de la versión 2000, vemos que las limitaciones son mucho más acusadas que en la versión 2005 y no solo a nivel de esta sintaxis, sino también en cuanto a su ejecución. Una de las mayores restricciones de esta versión es que no permite incluir la cláusula FOR XML en subconsultas [12], de forma que la única posición válida es justo al final de la consulta general. Afortunadamente, disponemos de la opción RAW y del argumento BINARY BASE64, que nos permitirán obtener información de forma similar al caso de la versión 2005 para casos muy concretos.

Extracción de ficheros usando serialización

El fundamento de la técnica descrita en este documento también es aplicable para otras técnicas de inyección existentes. Por ejemplo, si en lugar de serializar una tabla en formato XML, convertimos un fichero a su representación hexadecimal, podremos obtener su contenido íntegro con una sola petición al servidor:

- ```
  SELECT SYS.fn_VarBinToHexStr((SELECT c FROM OpenRowSet(BULK
  'c:\boot.ini', SINGLE_BLOB) AS T(c)))
  ```

Conclusiones

En este documento se ha descrito una nueva técnica de inyección SQL que permite la extracción de datos de una forma extremadamente eficiente usando serialización basada en XML. Si se compara la velocidad con las técnicas habituales utilizadas hasta el momento por las mejores herramientas disponibles actualmente, la diferencia resulta abismal (si bien es cierto que, al ser menos específicas, el ámbito de aplicación de dichas técnicas suele ser más amplio).

El procedimiento aquí descrito es automatizable casi en su totalidad, permitiendo la elaboración de herramientas que faciliten la extracción eficiente de datos, tal y como se describe en este mismo documento.


  ```

# Referencias


[1] **"NT Web Technology Vulnerabilities".** Rain Forest Puppy (1998)
http://www.wiretrip.net/rfp/txt/phrack54.txt

[2] **"Web Application Disassembly with ODBC Error Messages".** David Litchfield (2001)
http://www.nextgenss.com/papers/webappdis.doc

[3] **"Advanced SQL Injection in SQL Server Applications".** Chris Anley (2002)
http://www.ngssoftware.com/papers/advanced_sql_injection.pdf

[4] **"(more) Advanced SQL Injection".** Chris Anley (2002)
http://www.nextgenss.com/papers/more_advanced_sql_injection.pdf

[5] **"Blindfolded SQL injection".** Ofer Maor y Amichai Shulman (2003)
http://www.imperva.com/docs/Blindfolded_SQL_Injection.pdf

[6] **"Blind SQL Injection".** Kevin Spett (2003)
http://www.net-security.org/dl/articles/Blind_SQLInjection.pdf

[7] **"Automating Blind SQL Exploitation".** Cameron Hotchkies (2004)
http://althing.cs.dartmouth.edu/secref/resources/defcon12/dc-12-Hotchkies.ppt
http://www.0x90.org/releases/absinthe/

[8] **"FOR (cláusula de Transact-SQL)".** Microsoft MSDN Library, SQL Server 2005/2008.
http://msdn.microsoft.com/es-es/library/ms173812(SQL.90).aspx
http://msdn.microsoft.com/es-es/library/ms173812.aspx

[9] **"SQL Injection Cheat Sheet: Union Injections".** Ferruh Mavituna.
http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/#UnionInjections

[10] **"DataSet (Clase)".** Microsoft MSDN Library, .NET Framework
http://msdn.microsoft.com/es-es/library/system.data.dataset.aspx

[11] **"SELECT (Transact-SQL): FOR Clause".** Microsoft MSDN Library, SQL Server 2000.
http://msdn.microsoft.com/en-us/library/aa259187(SQL.80).aspx#_for_clause

[12] **"Guidelines for Using the FOR XML Clause".** MSDN Library, SQL Server 2000.
http://msdn.microsoft.com/en-us/library/aa226520(SQL.80).aspx