Departamento de
Automática y Computación
Automatika eta
Konputazio Saila

Campus de Arrosadía
Arrosadiko Campusa
31006 Pamplona - *Iruñea*
Tfno. 948 169113, Fax. 948 168924
Email: ayc@unavarra.es

Práctica 4: Seguridad en la web

1- Introducción

En esta práctica realizaremos, en primer lugar, un análisis de la seguridad de un sitio web empleando Nikto, uno de los analizadores de seguridad web más conocidos.

Posteriormente veremos el ataque por inyección SQL y sus correspondientes contramedidas.

2- Objetivos

Realizar de forma práctica algunos de los conceptos vistos en la teoría de seguridad en www efectuando ataques contra un servidor y analizando algunas de las herramientas disponibles como contramedidas.

3- Análisis de seguridad en WWW

Nikto es una herramienta que tiene como uso principal encontrar fallos de seguridad en un servidor web. Este programa en su configuración estándar busca:

- Fallos de configuración del servidor.
- Software no actualizado.
- Ficheros por defecto (que vienen con la instalación).
- Scripts o programas inseguros.

• Instalación de Nikto

Se puede descargar Nikto desde la URL: http://www.cirt.net/code/nikto.shtml

La instalación del programa consiste simplemente en descomprimir el software en un directorio de su home de prácticas:

1) Descargar el fichero `nikto-current.tar.gz`
2) Descomprimirlo con "`tar xvfpz nikto-current.tar.gz`"

• Análisis de la seguridad de un sitio web

El funcionamiento de Nikto es extremadamente sencillo:

```
$ nikto.pl -h <url> -output <url>_scan.txt
```

Con este comando Nikto lanzará sus análisis (llevará algún tiempo según la URL que escanee, pruebe con varias), devolviendo un informe con los resultados encontrados. Como referencia puede probar algunas de las listadas en:

http://securitythoughts.wordpress.com/2010/03/22/vulnerable-web-applications-for-learning/

Consulte la documentación de Nikto en `<directorio_nikto>/docs/nikto_manual.html`

Nota: Este programa sólo analiza los posibles fallos en la configuración de scripts, detecta versiones anticuadas de los programas más empleados y revisa la existencia de aplicaciones web

Campus de Arrosadía
Arrosadiko Campusa
31006 Pamplona - *Iruñea*
Tfno. 948 169113, Fax. 948 168924
Email: ayc@unavarra.es

muy usadas con vulnerabilidades conocidas. Nikto nunca nos podrá decir si nuestra aplicación web está programada de forma segura (la auditoría del código tendrá que realizarse manualmente).

Checkpoint 4.1: Analice el informe obtenido y describa las vulnerabilidades que ha encontrado en el sitio web. Referencias en http://osvdb.org/ ¿Sabría cómo explotar alguno de estos fallos de seguridad?

4- SQL injection y sus contramedidas

"SQL injection" es el nombre en inglés del ataque contra un Gestor de Bases de Datos Relacional(RDBM) que aprovecha la vulnerabilidad de una aplicación cliente del mismo. Dicha vulnerabilidad consiste en permitir mandar instrucciones SQL(Structured Query Language) adicionales a partir de un campo o un parámetro de entrada - por lo que se dice han sido "inyectadas". La finalidad del ataque es realizar tareas sobre la base de datos y de ser posible sobre el host mismo, teniendo resultados indeseables e inesperados que van desde la alteración de un dato hasta apoderarse del servidor.

El ataque "SQL injection" es posible dadas ciertas características del lenguaje SQL que lo dotan de flexibilidad, tales como:

- Poder embeber comentarios en una sentencia SQL.
- Poder escribir varias sentencias SQL juntas y ejecutarlas en bloque.
- Poder realizar consultas de metadatos por medio de "tablas de sistema".

Por este motivo cualquier RDBM que entiende SQL llámese Oracle, DB2, SQL Server, MySQL, etc. es susceptible de recibir un ataque de este tipo a través de sus aplicaciones cliente - este ataque se produce a nivel de aplicación - ya sean de tipo Consola, Windows o Web, independientemente de la plataforma de desarrollo (Java, .NET, etc.) con la que fue elaborada.

Escenario vulnerable.

Veamos un caso hipotético de una aplicación de acceso a datos que emplea entradas de usuario como parámetros de una consulta SQL común. Es típico que este tipo de consultas sean construidas dinámicamente utilizando sentencias SQL con concatenación de variables, al estilo:

```
"SELECT campo1,..., campoN FROM tablas WHERE campo1=" + mValor [+ ...]
```

Donde mValor esta dado por una entrada de usuario. Son estas entradas las puertas a un SQL injection ya que, dependiendo del tipo de dato de mValor, si en lugar de la entrada esperada se coloca:

a) ' or 1=’1 --

b) 0 or 1=1 --

c) #01/01/01# or 1=1 --

Campus de Arrosadía
Arrosadiko Campusa
31006 Pamplona - *Iruñea*
Tfno. 948 169113, Fax. 948 168924
Email: ayc@unavarra.es

Se puede realizar una consulta no deseada, ya que siguiendo el primer caso, el resultado es la sentencia siguiente:

```
"SELECT campo1,..., campoN FROM tablaX WHERE campo1='' Or 1='1' -- lo que siga no importa"
```

Lo que se consigue es hacer válida la consulta al añadir una cláusula OR que siempre será cierta (1=1) así como obligar al intérprete SQL a omitir el resto de la sentencia SQL original al introducir el guión doble (--) que le indica que lo que sigue es un comentario. Con ello el atacante puede, por ejemplo, tener acceso a la aplicación sin necesidad de contar con las credenciales adecuadas.

Por otro lado se podrían realizar consultas de los metadatos, es decir de las tablas del sistema del gestor, para conocer los nombres de las tablas de usuario y sus respectivos campos. Para ello podemos utilizar la cláusula UNION y hacer coincidir el número y los tipos de los campos de la consulta original de la aplicación para hacer algo como:

```
' UNION SELECT id, name, '', 0,'' FROM sysobjects WHERE xtype='U' --

' UNION SELECT 0, name, '', 0,'' FROM syscolumns WHERE id=[X] --
```

Con lo cual obtienen los nombres de las tablas de usuario y sus ID y a través de éstos últimos también sus columnas. Con dicha información es posible realizar cualquier tipo de consulta sobre la base de datos, limitados sólo por los privilegios de la cuenta con la que se realizan las consultas SQL en la aplicación.

Y con privilegios de administrador se puede ir mas allá, ya que algunos gestores como SQL Server tienen una serie de procedimientos almacenados que se pueden intentar ejecutar con esta técnica. Entre ellos existen algunos peligrosos como sp_OACreate y similares que permite la creación y manipulación de objetos ActiveX/COM y xp_cmdshell que permite ejecutar directamente comandos del sistema, por ejemplo:

```
'; EXEC xp_cmdshell 'net stop sqlserver', no_output
```

Con lo visto hasta aquí es suficiente para ilustrar la manera en que se realiza un ataque SQL Injection, así como su alcance. Pasemos ahora a la práctica.

Accedan a la página: http://www.greensql.net/sql-injection-test

1. Para la inyección SQL utilicen el campo “username” y como “Password” empleen su cuenta de prácticas (ssixy) o la cadena de texto que crea conveniente para que el ataque sea efectivo
2. Verifique diferentes ataques de inyección SQL y compruebe cómo lo detecta la herramienta de seguridad implementada entre el servidor web y el servidor donde se encuentra alojada la base datos de usuarios registrados (http://www.greensql.net/about).

Checkpoint 4.2: Muestre al profesor al menos 3 ataques “SQL injection” con éxito. ¿Qué consecuencias tendrían en un sistema sin protección?

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa

Departamento de
Automática y Computación
Automatika eta
Konputazio Saila

Campus de Arrosadía
Arrosadiko Campusa
31006 Pamplona - *Iruñea*
Tfno. 948 169113, Fax. 948 168924
Email: ayc@unavarra.es

Protegiéndonos de la SQL injection

Existen ciertos principios a considerar para proteger nuestras aplicaciones de un SQL injection:

1. *No confiar en la entrada del usuario.*
2. *No utilizar sentencias SQL construidas dinámicamente.*
3. *No utilizar cuentas con privilegios administrativos.*
4. *No proporcionar mayor información de la necesaria.*

A continuación veremos algunos ejemplos de como implementar dichos principios.

1. No confiar en la entrada del usuario significa:

- Filtrar la entrada del usuario de caracteres SQL para limitar los caracteres involucrados en un SQL Injection.

```
Private Function string SafeSqlLiteral( _
ByVal inputSQL As String) As String
Return inputSQL.Replace("'", "''")
End Function
'...
Dim safeSQL As String = SafeSqlLiteral(Login.Text)
Dim myCommand As SqlDataAdapter = _
New SqlDataAdapter("SELECT au_lname, au_fname " & _
"FROM authors WHERE au_id = '" & safeSQL & "'", _
myConnection)
```

- Proteger las instrucciones de búsqueda de modelos coincidentes (LIKE).

```
Private Function SafeSqlLikeClauseLiteral( _
ByVal inputSQL As String) As String
Dim s As String = inputSQL
s = inputSQL.Replace("'", "''")
s = s.Replace("[", "[[]")
s = s.Replace("%", "[%]")
s = s.Replace("_", "[_]")
Return s
End Function
```

- Verificar tanto el tamaño como el tipo de los datos de las entradas de usuario:
  - Evitando los siguientes tipos caracteres de riesgo para el gestor de datos:
    - El delimitador de consultas: Punto y coma (;)
    - Delimitador de datos tipo cadena de caracteres: Comilla sencilla (').
    - Delimitadores de cometario: Guión doble (--) y "/*..*/" en el caso de SQL Server.

Nota: En lugar de evitar los caracteres peligrosos, otro modo de protegernos es aceptar sólo los caracteres inofensivos.

  - Evitando las cadenas con el inicio de nombres de las tablas y los procedimientos del sistema: "sys" y "xp_" en el caso de SQL Server. Así como las siguientes palabras: AUX, CLOCK$, COM1, COM8, CON, CONFIG$, LPT1, LPT8, NUL y PRN
  - Utilizar preferentemente controles con valores predefinidos o discretos tales como cuadros de lista, cuadros combinados, de verificación, etc. en lugar de cuadros de texto.
- Verificar cualquier tipo de entrada, no sólo lo introducido en los controles IU sino también aquellas que no son visibles, como parámetros de entrada y campos tipo hidden de las páginas web.

Realizar la verificación en todos los niveles y capas de la aplicación, ya que si sólo protegemos la capa de presentación somos vulnerables a que un atacante salte a la siguiente capa y realice su ataque.

2. No utilizar sentencias SQL construidas dinámicamente. En lugar de ello:

- Utilizar instrucciones SQL con Parámetros.

```
Dim myCommand As SqlDataAdapter = New SqlDataAdapter( _
"SELECT au_lname, au_fname FROM Authors " & _
"WHERE au_id= @au_id", myConnection)
Dim parm As SqlParameter = _
myCommand.SelectCommand.Parameters.Add("@au_id", _
SqlDbType.VarChar, 11)
parm.Value = Login.Text
```

Aunque de hecho es mejor utilizar procedimientos almacenados siempre que sea posible, *así como también*:

- Utilizar parámetros al llamar a procedimientos almacenados.

```
Dim myCommand As SqlDataAdapter = _
New SqlDataAdapter("AuthorLogin", myConnection)
myCommand.SelectCommand.CommandType = _
CommandType.StoredProcedure
Dim parm As SqlParameter = _
myCommand.SelectCommand.Parameters.Add("@LoginId", _
SqlDbType.VarChar,11)
parm.Value = Login.Text
```

3. No utilizar cuentas con privilegios administrativos.

- Ejecutar las sentencias SQL o invocar Procedimientos Almacenados con una cuenta con privilegios mínimos. Nunca emplear 'sa' en el caso de MS SQL Server.
- Conceder permisos de ejecución únicamente a Procedimientos Almacenados propios desde los cuales, a manera de "wraper", se realicen las consultas a las Tablas de Usuario y llamadas a los Procedimientos Almacenados del Sistema que se requieran en las aplicaciones, y negar el acceso directo a éstos últimos y a las Tablas de Usuario.

Campus de Arrosadía
Arrosadiko Campusa
31006 Pamplona - *Iruñea*
Tfno. 948 169113, Fax. 948 168924
Email: ayc@unavarra.es

4. No proporcionar mayor información de la necesaria.

- No exponer al usuario final los mensajes de error devueltos por el gestor de la base de datos, para no brindar mayor información que sea útil al atacante.
- Implementar un sistema de gestión de errores que notifique del mismo únicamente a los administradores de la aplicación y el gestor de la base de datos. Por ejemplo, para el caso de una aplicación web, establecer en el archivo de configuración web (Web.Config) el valor del atributo debug del elemento compilation en False y el atributo mode del elemento customErrors en On o en su defecto en RemoteOnly.

```
<compilation defaultLanguage="vb" debug = "false" />
<customErrors mode="RemoteOnly"
defaultRedirect="GenericErrorPage.htm">
</customErrors>
```

Sin embargo para entornos de producción donde la variedad de usuarios y aplicaciones disponibles llegan a escapar del control del administrador, es necesario aplicar medidas adicionales con las que evitar la intrusión aún existiendo vulnerabilidad a la inyección SQL.

Un ejemplo de estas contramedidas lo acaban de comprobar en el apartado anterior.

Checkpoint 4.3: Muestre al profesor de prácticas en qué consiste esta protección activa y cuál es su arquitectura. Encuentre al menos un par más de contramedidas software para entornos sensibles a la SQL injection.

Encuentre al menos 3 scanners SQL injection que le permitan comprobar si un sitio web es vulnerable a la inyección SQL.

Conclusión

Para prevenir un ataque SQL injection, así como para realizar pruebas de vulnerabilidad contra el mismo en nuestras aplicaciones, no debemos olvidar que cualquier aplicación que permita una "entrada" que sirva de parámetro para una consulta SQL es vulnerable a este ataque.