Bug Report Link:
http://bugs.mysql.com/bug.php?id=169

Buggy Software Version:
mysql 3.23.56

Program description:
Open source Database Server
Source code download link xxx

Bug Search Keyword:
race

Bug root-cause explanation:
Type: Multi-Variable Atomicity violation

sql_delete.cc; in function generate_table

```
int generate_table(THD *thd, TABLE_LIST *table_list, TABLE *locked_table)
{
  //table generation work
  ...
  mysql_update_log.write(...) //line 109
  ...
}
```

In this function, MySQL first generates a table. This generation itself is done with good locking, checking, etc.

After the table generation, MySQL releases ALL the locks held during table generation, and then tries to log this table generation action into binlog (the database log used for recovery,etc).

Note that, the table generation and the logging is NOT put in any critical section. Therefore they may be interleaved by other binlog updating. As a result, the binlog may not match the actual database execution order, which would cause both security and potential availability issues.

For example, another thread may be executing table delete.
The database operation order maybe: table is generated, followed by the table is deleted.
However, the database log may be showing table deletion first, followed by table generation.

How to trigger the bug

1) Input

Two database requests. One generates a table; the other delete the same table.

```
> ./mysql -u root -D test -e 'delete from b' &
> ./mysql -u root -D test -e 'insert into b values (1)' &
```

2) Timing

Adding 'sleep(x)' right before the mysql_update_log.write (around line 108) can make the bug

easily repeated.

Errors & Failures:
There is no explicit errors, crashes, exceptions, or anything.

The failure is that the log does not match with the "real" execution order. This is a "silent" failure.

How is this bug fixed by developers?
Developers extend an existing lock critical section to put mysql_update_log.write into the same critical section as table generation.

How automated tools would behave on this bug:

1) bug detection

   This atomicity bug involves actions on two different objects (mysql database table object; mysql log).
   We expect this to be very difficult to detect by traditional race detectors.

   This bug would be missed by race detection, AVIO.

   We hoped that it can be detected by MUVI. Unfortunately, MUVI also missed this bug because it involves `conditional' variable correlation.

2) failure diagnosis

   This bug will be difficult to diagnose, because its failure is a silent failure.

3) failure prevention/recovery

   Failure recovery will be difficult, because of the silent failure problem.
   Existing failure prevention technique (e.g., AI.FSE14) would not work for this bug, because it involves more than one variable.

4) bug fixing

   This bug could be automatically fixed by moving lock/unlock, as long as it can be automatically detected.