



# SQL injection

What is it and how to avoid it



PERCONA  
LIVE

# WHAT IS SQL INJECTION?

# What is SQL injection?

6

- SQL injection is an attack vector
  - An attacker modifies the SQL queries which will be executed by the server
  - But the attacker *does not need to change the code on the server or get access to the server*



# What is SQL injection – interpolation (strings)

7

```
$username = $_GET['username'];  
$sql =  
"select 1  
  from users.users  
 where admin_flag=true  
   and username = " . $username . " " ;
```

\$ wget http://host/path.php?username=bob

SQL injection!

\$ wget http://host/path.php?user\_id=" or '1'='1"

❑ **and username = ' or '1' = '1'**

# Escape strings, or use prepared statements!

8

#escape string values

```
$username = mysqli_real_escape_string($_GET['username']);  
$sql = "select ... and username = " . $username . " ";
```

#prepared statement

```
$username = GET['username'];  
$stmt = mysqli_stmt_init($conn);  
$sql = "select ... and username = ?"  
mysqli_stmt_prepare($stmt, $sql);  
mysqli_stmt_bind_param($stmt, "s", $username);  
mysqli_stmt_execute($stmt);  
mysqli_stmt_close($stmt);
```

# What is SQL injection – interpolation (ints)

9

```
$user_id = $_GET['user_id'];  
$sql =  
"select 1  
  from users.users  
 where admin_flag=true  
   and user_id = " . $user_id;  
...  
  
$ wget http://host/path.php?user_id=1  
$ wget http://host/path.php?user_id="1 or 1=1"
```

SQL injection!



# Use type checking, or prepared statements!

10

```
#check that integers really are integers!  
$user_id = GET['user_id'];  
if(!is_numeric(user_id)) $user_id = "NULL";  
$sql = "select ... and user_id = " . $user_id;
```

```
#prepared statement  
$user_id = GET['user_id'];  
$sql = "select ... and user_id = ?"  
...  
mysqli_stmt_bind_param($stmt, "i", $user_id);  
mysqli_stmt_execute($stmt);
```



# When escaping can't help

11

- Some parts of a SQL statement can't be manipulated using parameters
- These include
  - ORDER BY columns
  - Variable number of items in an IN list
  - Adding SQL syntax like DISTINCT



# Don't use user input in the query

12

```
#avoid using user input directly in ANY way
$sql = "select * from listings where deleted = 0 and sold
= 0 and open = 1";
if(!empty($_GET['ob'])) {
    $sql .= " ORDER BY " . $_GET['ob'];
}
```

Bad!

wget ... ?ob=post\_date


wget ... ?ob="post\_date union all (select \* from listings)"

Now we can see all listings

# Use whitelisting instead

13

```
#avoid using user input directly in ANY way
$sql = "select * from listings where deleted = 0 and sold
= 0 and open = 1";
$allowed = array('post_date','neighborhood','etc');
if(!empty($_GET['ob']) && is_string($_GET['ob'])) {
    if(in_array($_GET['ob'], $allowed)) {
        $sql .= " ORDER BY " . $_GET['ob'];
    }
}
```

 in\_array() is the keeper of the gate

wget ... ?ob=post\_date

wget ... ?ob="post\_date union all (select \* from listings)"

# All that works great for the apps you control

14

- **BUT...**
  - If you don't have the source for an app, then you really can't be sure it isn't safe from SQL injection
  - Or maybe you have to support old apps
  - Or apps that were not developed rigorously
  - **What do we do in these cases?**

Out-of-band SQL injection detection

# SQL INJECTION **DETECTION** USING PT-QUERY-DIGEST



# How to detect SQL injection?

16

- Most applications only do a small number of things.
  - Add orders, mark orders as shipped, update addresses, etc.
  - The SQL “patterns” that identify these behaviors can be collected and whitelisted.
  - Queries that don’t match a known fingerprint may be investigated as SQL injection attempts

# What is a query fingerprint?

17

- A query fingerprinting algorithm transforms a query into a form that allows like queries to be grouped together and identified as a unit
  - In other words, these like queries share a fingerprint
  - Even though the queries differ slightly they still fingerprint to the same value
  - This is a heuristic based approach

# Tools that support query fingerprints

18

- Percona Toolkit tools

- pt-query-digest ←

Reads slow query logs and populates the whitelist table.

Can also be used to display new queries that have not been marked as allowed.

- pt-fingerprint ←

Takes a query (or queries) and produces fingerprints.

Useful for third party tools that want to use fingerprints.

## What is a query fingerprint (cont?)

19

`select * from some_table where col = 3`

becomes

`select * from some_table where col = ?`

`select * from some_table where col = IN (1,2)`

becomes

`select * from some_table where col IN (?)`



# Query fingerprints expressed as hashes

20

pt-query-digest can provide short hashes of checksums

```
select * from some_table where col = ?  
982e5737f9747a5d (1631105377)
```

```
select * from some_table where col = IN (?)  
2da8ed487cdfc1c8 (1680229806268)
```

base 10

- Normally used for profiling slow queries
- Has a “SQL review” feature for DBAs
  - Designed to mark query fingerprints as having been reviewed
  - This feature can be co-opted to discover new query fingerprints automatically
  - New fingerprints are either new application code or SQL injection attempts

## pt-query-digest – review feature

22

- Need to store the fingerprints in a table
  - Known good fingerprints will be marked as reviewed
  - If pt-query-digest discovers new fingerprints you will be alerted because there will be unreviewed queries in the table

# pt-query-digest - review table initialization

23

## Need to initialize the table

```
pt-query-digest /path/to/slow.log \
```

```
--create-review-table
```

```
--review "h=127.0.0.1,P=3306,u=percona,p=2un1c0rns,D=percona,t=whitelist" \
```

```
--sample 1 \
```

```
--no-report
```

Where to store fingerprints

Don't waste time on stats

Don't print  
report



# pt-query-digest – command-line review

24

pt-query-digest /path/to/slow.log \

--review "DSN..." \ ← How it knows which queries have already been reviewed

--sample 1 \ ← Don't collect stats, just sample one of each new fingerprint

--report \ ← Display the report of queries

--limit 0 ← Ensure that all unreviewed queries are shown

# USING THE WHITELIST WITH SQL

# Detecting new query fingerprints

26

```
SELECT count(*)  
  FROM percona.whitelist  
 WHERE reviewed_by IS NULL;
```

Any new queries?  
percona.whitelist is just  
an example name, you can  
use any you like

```
SELECT checksum, sample  
  FROM percona.whitelist  
 WHERE reviewed_by IS NULL;
```

Get a list of the  
queries

# Add a query fingerprint to the whitelist

27

```
UPDATE percona.whitelist  
    SET reviewed_by = 'allow',  
        reviewed_on = now()  
WHERE checksum= 1680229806268;
```



Out of band detection

# **LIMITATIONS AND CAVEATS**

# Out-of-band detection

31

- Some damage or information leakage may have already happened
- To limit the extent of the damage send an alert as soon as a new pattern is detected
  - Ensure thorough application pattern detection in a test environment to avoid false positives

# Get logs as fast as possible

32

- Use tcpdump on a mirrored server port
  - Pipe the output to pt-query-digest
- Use tcpdump on the database server
  - Adds some additional overhead from running the tools on the same machine
  - Possibly higher packet loss
- Collect and process slow query logs frequently
  - Adds slow query log overhead to server
  - Longer delay before processing

What to do **BEFORE** a fishy fingerprint appears

# FINDING THE VULNERABILITY



## Prepare for finding a vulnerability

34

- Tracking down the vulnerable code fragment can be difficult if you have only the SQL statement
- Not just a problem with SQL injection since it is usually convenient to see where a SQL statement was generated from

# Add tracing comments to queries

35

- A good approach is to modify the data access layer (DAL) to add SQL comments
  - Comments are preserved in the slow query log
  - Comments are displayed in SHOW commands
    - SHOW ENGINE INNODB STATUS
    - SHOW PROCESSLIST
  - Make sure your client does not strip comments!

## Add tracing information

36

- PHP can use `debug_backtrace()` for example
- PERL has variables that point to the file and line
- Investigate the debugging section of your language's manual

# What to place in the comment

37

- Here are some important things to consider placing into the tracing comment
  - session\_id (or important cookie info)
  - application file name, and line number
  - important GET, POST, PUT or DELETE contents
  - Any other important information which could be useful for tracking down the vector being used in an attack



# Example comments in SQL queries

38

```
select airport_name, count(*)  
  from dim_airport  
  join ontime_fact  
    on dest_airport_id = airport_id  
 where depdelay > 30  
    and flightdate_id = 20080101
```

```
/*  
webserver:192.168.1.3,file:show_delays.php,1  
ine:326,function:get_delayed_flights,user:ju  
stin,sessionid:7B7N2PCNIOKCGF  
*/
```

This comment contains all that you need

# Most apps don't do this out of the box

39

- You can modify the application
  - If you have the source code (and it uses a DAL)
- BUT...
  - There isn't much you can do if
    - The application is closed source , or you can't change the source
    - There is no DAL (code/query spaghetti)
    - For any other reason it is problematic to inject information into all SQL queries

# If I can't change the source?

40

- You can't fix the problems when you detect them.
- Consider using an open source solution
- Or consider in-band protection

Noinject-MySQL – My Lua script for MySQL proxy + web interface

GreenSQL – Commercial proxy for MySQL and other databases

MySQL Enterprise Firewall – New SQL injection prevention firewall from Oracle

# Q/A