Preventing SQL and PL/SQL Injection Attacks

Arup Nanda

Longtime Oracle Technologist

DocID: P91801 Date: 20161104

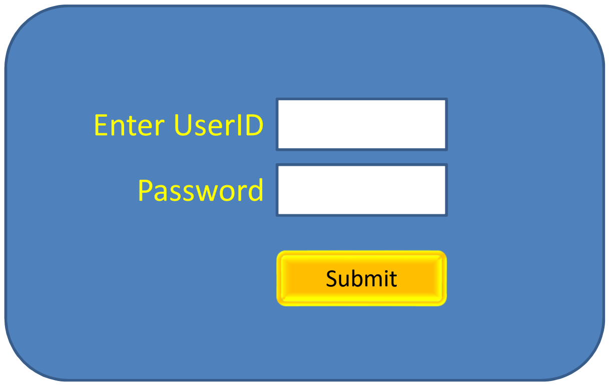
Why This Session?

- CIO Survey
  - What keeps you up at night?
- Results
  - Used to be: database is down
  - Now: Data breach
- SQL Injection Attacks are growing.

Agenda

- What is SQL Injection
- Different types of injection
- How to protect the database
- How to write bulletproof code

Entering forms



Checking Passwords

- A Table Called USER_ACCESS

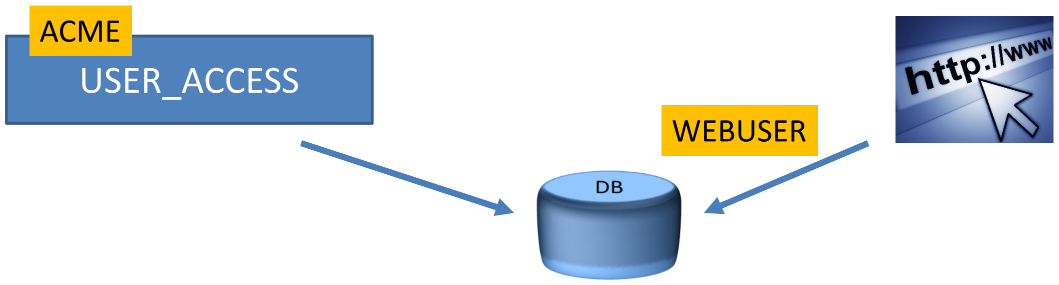
```
SQL> desc user_Access
 Name        Null?    Type
 -------- ------- ------------
 USERNAME          VARCHAR2(30)
 PASSWORD          VARCHAR2(30)
```

- Check Password

```
select password
from user_access
where username = 'ARUP';
```

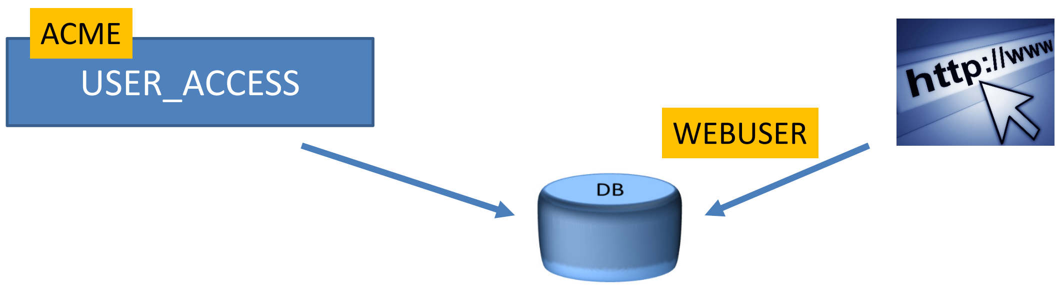
Two Users

- Table USER_ACCESS should be owned by a privileged schema
- Website user should be a different one, e.g. WEBUSER



First Problem

- The WEBUSER should have SELECT privileges on USER_ACCESS



Create a Function (as ACME)

```
create or replace function is_password_correct (
   p_username in varchar2,
   p_password in varchar2
) return varchar2 is
   l_stmt    varchar2(4000);
   l_check   varchar2(10) := 'wrong';
begin
   l_stmt := 'select ''correct'' from user_access '||
     'where username = '''||
     p_username ||''' and password = '''
     ||p_password||'''';
   dbms_output.put_line('l_stmt='||l_stmt);
   execute immediate l_stmt into l_check;
   return l_check;
end;
```

func1

Inject SQL

```
select is_password_correct('ARUP','ARUPPASS') from dual                check0
                                                                       check1

select is_password_correct('ARUP','wrongpass'' or ''1''=''1') from dual  check2
```

$\underbrace{\texttt{'' or ''1''=''1}}_{\textbf{\textit{Injected String}}}$

Various Checkers

- Input `'wrongpass'' or ''1''=''1'`
- Check 1=1 in the input script
- Could be anything
  - `‘’2’’=‘’2’`
  - `‘banana’ = ‘banana’`
- Checkers can’t be exhaustive.

Bind Variable

```
create or replace function is_password_correct (
  p_username in varchar2,
  p_password in varchar2 )
return varchar2 is
  l_stmt           varchar2(4000);
  l_check          varchar2(10) := 'wrong';
begin
  l_stmt :=     'select ''correct'' from user_access '||
     'where username = '''||
      p_username ||''' and password = :l_password';
     dbms_output.put_line('l_stmt='||l_stmt);
     execute immediate l_stmt into l_check
     using p_password;
     return l_check;
end;
/
```

func2

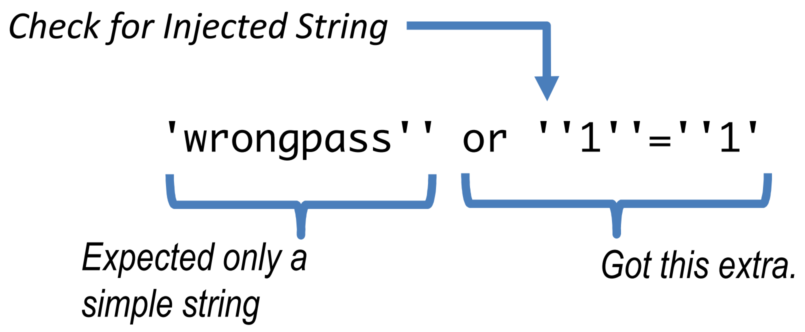
check1

Bind Variables

- Great for
  - Security
  - Parsing
- Drawbacks
  - Extensive change required

```
p_username ||''' and password = :l_password';
dbms_output.put_line('l_stmt='||l_stmt);
execute immediate l_stmt into l_check
using p_password;
```

Assertion


Check for Injected String
'wrongpass'' or ''1''=''1'
Expected only a simple string
Got this extra.

DBMS_ASSERT

- Check the input string:
```
select dbms_assert.enquote_literal (‘WrongPass’)
from dual;
```
assert1
- Injected string:
```
select dbms_assert.enquote_literal ('WrongPass'' or
''1''=''1') from dual
```
assert2

Modified Function

```
create or replace function is_password_correct (
   p_username in varchar2,
   p_password in varchar2
) return varchar2 is
   l_stmt    varchar2(4000);
   l_check   varchar2(10) := 'wrong';
begin
   l_stmt := 'select ''correct'' from user_access '||
     'where username = '''||
     p_username ||''' and password = '''
     ||~~p_password~~||'''';  sys.dbms_assert.enquote_literal(p_password)
   dbms_output.put_line('l_stmt='||l_stmt);
   execute immediate l_stmt into l_check;
   return l_check;
end;
```

func3

check1

Problem

- Message is not clear

```
select dbms_assert.enquote_literal('WrongPass'' or
       *
ERROR at line 1:
ORA-06502: PL/SQL: numeric or value error
ORA-06512: at "SYS.DBMS_ASSERT", line 409
ORA-06512: at "SYS.DBMS_ASSERT", line 493
```

Modified Function

```
begin
   declare
     l_possible_injection_exception exception;
     pragma exception_init (l_possible_injection_exception,-6502);
   begin
     l_sanitized_password := sys.dbms_assert.enquote_literal(p_password);
   exception
     when l_possible_injection_exception then
       raise_application_error (-20001,'Possible SQL Injection Attack');
     when OTHERS then
       raise;
   end;
   l_stmt :=
     'select ''correct'' from user_access where username = '''||
      p_username ||''' and password = '||l_sanitized_password||'';
   dbms_output.put_line('l_stmt='||l_stmt);
   execute immediate l_stmt into l_check;
   return nvl(l_check,'wrong');
 end;
```

func4

check2

Another Injection

- Create ACCOUNTS Table
- Identify the minimum balance from the list of account numbers passed

```
create or replace function get_min_bal_acclist
(
   p_acc_list in varchar2
)
return number
is
   l_stmt  varchar2(32000);
   l_ret   number;
begin
   l_stmt := 'select min(principal) from accounts where acc_no in ('
           ||p_acc_list||')';
   dbms_output.put_line ('l_stmt='||l_stmt);
   execute immediate l_stmt into l_ret;
   return l_ret;
end;
```

f1

q1

Injection

- Normal Input

  `get_min_bal_acclist('101,203,345')`

q1

- Injected Input

  `get_min_bal_acclist('''9999'') or (1=1')`

q2

Protect by a Type

- Pass the list as a collection; not as a string
- Create a type

```
create type ty_acc_no_list is table of number;
/
```

cr_ty_acc_no_list

Modified Function

```
create or replace function get_min_bal_acclist
(
   p_acc_list in ty_acc_no_list
)
return number
is
   l_ret   number;
begin
   select min(principal)
   into l_ret
   from accounts
   where acc_no in (
      select /*+ dynamic_sampling (t1 2) */ column_value
      from table (p_acc_list) t1
   );
   return l_ret;
end;
/
```

f2
q2

Assert Properties

- Puts Single Quotes
- Example 1
  - `select dbms_assert.enquote_literal('WrongPass') from dual`
  - Output:
  - `'WrongPass'`
- Example 2
  - `select dbms_assert.enquote_literal('''WrongPass''') from dual`
  - Output:
  - `'WrongPass‘`
- Double quotes have no impact

Date Injection

Pipelined Function to Get Age

cr_get_recently_opened_accno

```
create or replace function get_recently_opened_accno
(
   p_created_after_dt      in date
)
return ty_acc_no_list
pipelined
as
   l_acc_no_list    ty_acc_no_list;
   l_stmt          varchar2(4000);
begin
   l_stmt := 'select acc_no from accounts where created_dt > '''||
           p_created_after_dt||''' order by acc_no';
   dbms_output.put_line('l_stmt='||l_stmt);
   execute immediate l_stmt
   bulk collect into l_acc_no_list;
   for ctr in l_acc_no_list.first..l_acc_no_list.last loop
           pipe row (l_acc_no_list(ctr));
   end loop;
   return;
end;
```

Inject

- As WEBUSER:

```
select * from table(acme.get_recently_opened_accno('01-NOV-15'));
```

q6

q7

- As WEBUSER:

```
alter session set nls_date_format= '"'' or ''1''=''1"'
/
select * from table(acme.get_recently_opened_accno(to_date('01-OCT-16','dd-
mon-rr')))
/
```

Date Format Transformation

```
alter session set nls_date_format= '"'' or ''1''=''1"'

created_dt > '01-NOV-15'  →  created_dt > '' or '1'='1'
```

Prevention

cr_get_recently_opened_accno2

```
create or replace function get_recently_opened_accno
   (p_created_after_dt in date)
return ty_acc_no_list
pipelined
as
   l_acc_no_list    ty_acc_no_list;
   l_stmt          varchar2(4000);
begin
   l_stmt := 'select acc_no from accounts where created_dt > '''||
           to_char(p_created_after_dt,'DD-MON-RR')||
           ''' order by acc_no';
   execute immediate l_stmt
   bulk collect into l_acc_no_list;
   for ctr in l_acc_no_list.first..l_acc_no_list.last loop
           pipe row (l_acc_no_list(ctr));
   end loop;
   return;
end;
```

Effect

- As WEBUSER:

```
select * from table(acme.get_recently_opened_accno(to_date('01-OCT-16','dd-
mon-rr')))
                    *
ERROR at line 1:
ORA-01861: literal does not match format string
ORA-06512: at "ACME.GET_RECENTLY_OPENED_ACCNO", line 15
```

Anonymous Block

Anonymous Block

anon1

```
declare
   procedure printf
   (
           p_input in varchar2
   )
   is
           l_stmt  varchar2(32767);
   begin
           l_stmt := 'begin dbms_output.put_line(''LOG ['||
                   to_char(sysdate,'mm/dd/yy-hh24:mi:ss') ||'] '||
                           p_input||
                   '''); end;';
           execute immediate l_stmt;
           -- dbms_output.put_line('l_stmt='||l_stmt);
   end;
begin
   printf ('Starting the process');
   -- some activity occurs here
   printf ('Inbetween activities');
   -- some more activities
   printf ('Ending the process');
end;
```

Injected Block

anon2

```
declare
   procedure printf
   (
           p_input in varchar2
   )
   is
           l_stmt  varchar2(32767);
   begin
           l_stmt := 'begin dbms_output.put_line(''LOG ['||
                   to_char(sysdate,'mm/dd/yy-hh24:mi:ss') ||'] '||
                           p_input||
                   '''); end;';
           execute immediate l_stmt;
           -- dbms_output.put_line('l_stmt='||l_stmt);
   end;
begin
   printf ('Starting the process');
   -- some activity occurs here
   printf ('Inbetween activities');
   -- some more activities
   printf ('Ending the process''); execute immediate ''grant select on accounts to public''; end; --');
end;
```

Prevention

anon3

```
declare
   procedure printf
   (
           p_input in varchar2
   )
   is
           l_stmt  varchar2(32767);
           l_temp  varchar2(32767);
   begin
           l_temp := dbms_assert.enquote_literal(p_input);
           l_stmt := 'begin dbms_output.put_line(''LOG ['||
                   to_char(sysdate,'mm/dd/yy-hh24:mi:ss')||'] '''||'||
                           l_temp||
                   '); end;';
           dbms_output.put_line('l_stmt='||l_stmt);
           execute immediate l_stmt;
   end;
begin
   printf ('Starting the process');
   -- some activity occurs here
   printf ('Inbetween activities');
   -- some more activities
   printf ('Ending the process''); execute immediate ''grant select on accounts to public''; end; --');
end;
```

Other Assertion Procedures

Valid Name?

- Check for Valid Identifiers

```
select dbms_assert.enquote_name('One'1Day') from dual;
```

assert4

- Puts double quotes
- But if double quotes exist, then it doesn't put once again

```
select dbms_assert.enquote_name('"1Day"') from dual
```

assert5

SQL Name

- Is it a valid name for an SQL name?

```
select dbms_assert.qualified_sql_name('Arup') from dual;
```

- Invalid Name

```
select dbms_assert.qualified_sql_name('1rup') from dual;
```

- Returns value without quotes

```
select dbms_assert.qualified_sql_name('A''rup') from dual
       *
ERROR at line 1:
ORA-44004: invalid qualified SQL name
ORA-06512: at "SYS.DBMS_ASSERT", line 315
```

Valid Schema?

- Is this an existing schema name?

```
SQL> select dbms_assert.schema_name('HR') from dual;

DBMS_ASSERT.SCHEMA_NAME('HR')
-----------------------------
HR

SQL> select dbms_assert.schema_name('H1R') from dual;
select dbms_assert.schema_name('H1R') from dual
       *
ERROR at line 1:
ORA-44001: invalid schema
ORA-06512: at "SYS.DBMS_ASSERT", line 333
```

Multiple Object Injection

Function to Get # of Rows

```
create or replace function get_total_rows
(
   p_table_name    in user_tables.table_name%type
)
return number
is
   l_cnt   number;
   l_stmt  varchar2(2000);
begin
   l_stmt := 'select count(*) from '||
           p_table_name;
   dbms_output.put_line ('l_stmt='||l_stmt);
   execute immediate l_stmt into l_cnt;
   return l_cnt;
end;
```

cr_get_total_rows1

Injection

- Normal use:

```
select acme.get_total_rows('ACCOUNTS') from dual;
```

q9

- Injection:

```
select acme.get_total_rows('ACCOUNTS,CREDIT_CARDS') from dual;
```

q10

Prevention

```
create or replace function get_total_rows
(
   p_table_name    in user_tables.table_name%type
)
return number
is
   l_cnt   number;
   l_stmt  varchar2(2000);
begin
   l_stmt := 'select count(*) from '||
           sys.dbms_assert.sql_object_name(p_table_name);
   dbms_output.put_line ('l_stmt='||l_stmt);
   execute immediate l_stmt into l_cnt;
   return l_cnt;
end;
```

cr_get_total_rows2

Tips for Avoiding SQL Injection Attacks

Avoid Dynamic SQL

- Static

```
select count(*)
into l_count
from accounts;
```

- Dynamic

```
l_stmt := 'select count(*) from accounts';
execute immediate l_stmt into l_cnt;
```

Use Binds Over Concat

- Concatenation

```
l_stmt := 'select ''true'' from user_access where
username = ''SUPERUSER'' and password =
'''||p_password||'''';
```

- Binds

```
l_stmt := 'select ''true'' from user_access where
username = ''SUPERUSER'' and password = :l_password';
```

Use a Password Check Function

```
create or replace function
password_is_correct
(
   p_username      in
user_access.username%type,
   p_password      in
user_access.password%type
)
return boolean
as
   l_password      user_access.password%type;
begin
   select password
   into l_password
   from user_access
   where username = p_username;
   if (l_password = p_password) then
           return true;
   else
           return false;
   end if;
```

```
begin
   if password_is_correct
('SUPERUSER','SuperPass') then
      dbms_output.put_line ('Password is
correct');
   else
       dbms_output.put_line ('Either
Userid or Password is NOT correct');
   end if;
end;
```

Date Format

- Do not use this

```
l_stmt :=
    'select accno from accounts where created_dt > '''||
            p_created_after_dt||
            ''' order by accno';
```

- Use this instead:

```
l_stmt :=
    'select accno from accounts where created_dt > '''||
            to_char(p_created_after_dt,'DD-MON-RR')||
            ''' order by accno';
```

Use DBMS_ASSERT for validating and sanitizing inputs

Thank You!

Blog: arup.blogspot.com *Download this session here.*
Tweeter: @ArupNanda
Facebook.com/ArupKNanda