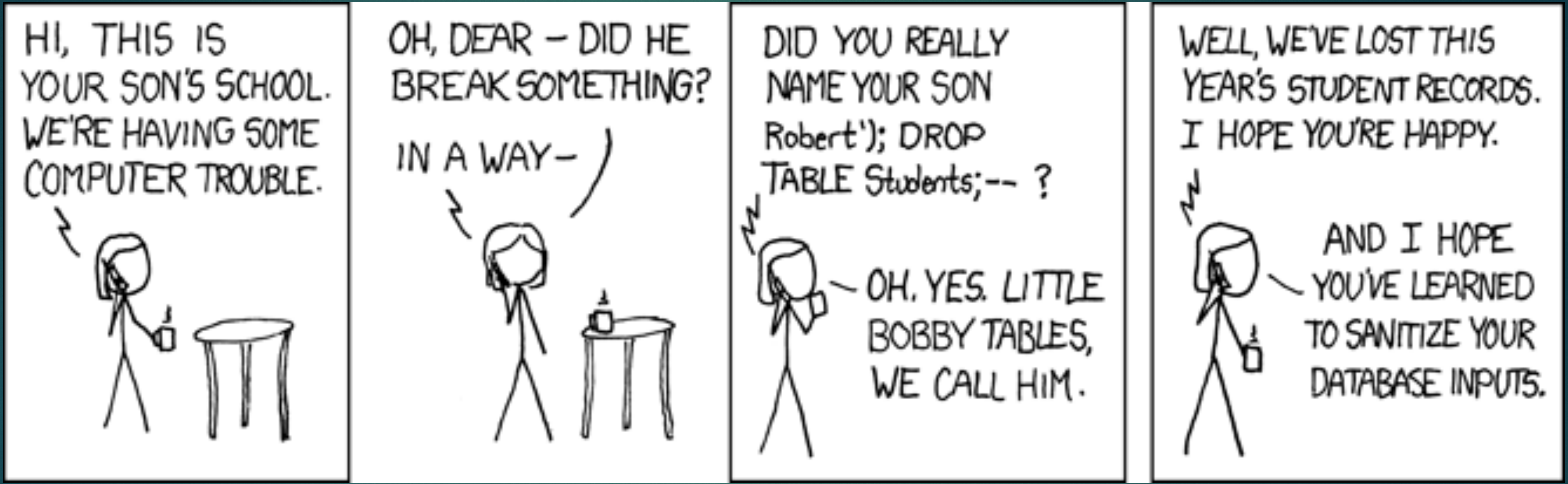
SQL Injection – What is it?

- Inputs to the application that interact with a backend database directly can be vulnerable to SQL injection.
- Developers can be too trusting and some don't realise that SQL queries can be tampered with.
- An attacker sends a complete or partial SQL query via an input to the application, e.g, interacting with login box in a web browser and sending that to the web application.

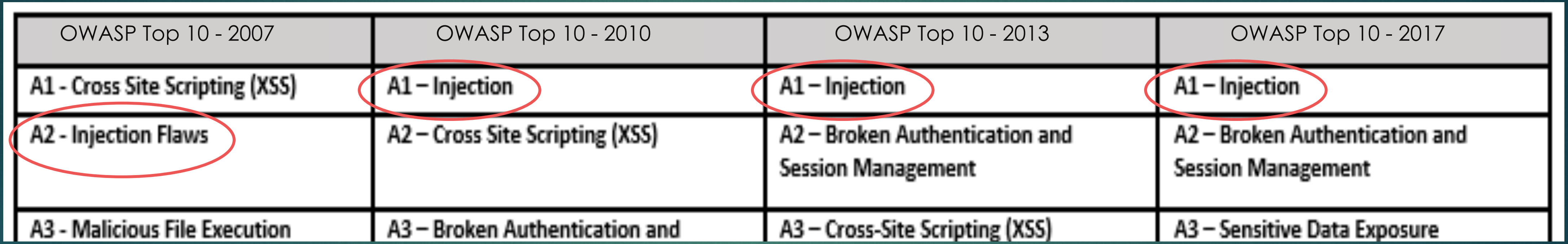
SQL Injection – What is it?


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

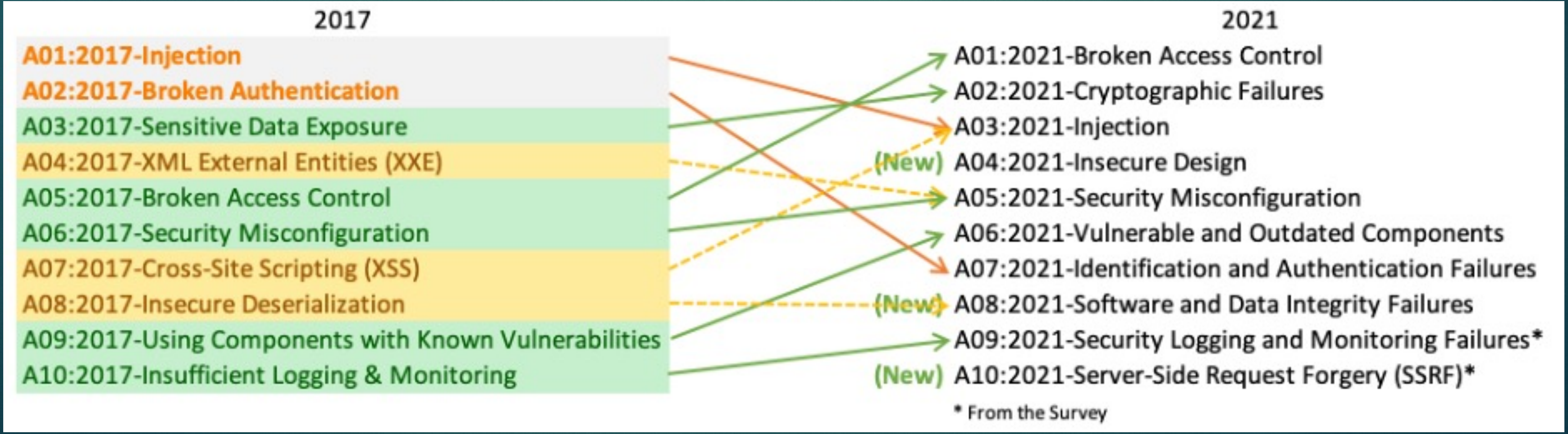
https://xkcd.com/327/

SQLi in Industry

- Consistently towards the top of OWSAP top 10



OWASP Top 10 - 2007	OWASP Top 10 - 2010	OWASP Top 10 - 2013	OWASP Top 10 - 2017
A1 - Cross Site Scripting (XSS)	A1 – Injection	A1 – Injection	A1 – Injection
A2 - Injection Flaws	A2 – Cross Site Scripting (XSS)	A2 – Broken Authentication and Session Management	A2 – Broken Authentication and Session Management
A3 - Malicious File Execution	A3 – Broken Authentication and	A3 – Cross-Site Scripting (XSS)	A3 – Sensitive Data Exposure



2017

- A01:2017-Injection
- A02:2017-Broken Authentication
- A03:2017-Sensitive Data Exposure
- A04:2017-XML External Entities (XXE)
- A05:2017-Broken Access Control
- A06:2017-Security Misconfiguration
- A07:2017-Cross-Site Scripting (XSS)
- A08:2017-Insecure Deserialization
- A09:2017-Using Components with Known Vulnerabilities
- A10:2017-Insufficient Logging & Monitoring

2021

- A01:2021-Broken Access Control
- A02:2021-Cryptographic Failures
- A03:2021-Injection
- (New) A04:2021-Insecure Design
- A05:2021-Security Misconfiguration
- A06:2021-Vulnerable and Outdated Components
- A07:2021-Identification and Authentication Failures
- (New) A08:2021-Software and Data Integrity Failures
- A09:2021-Security Logging and Monitoring Failures*
- (New) A10:2021-Server-Side Request Forgery (SSRF)*

* From the Survey

SQLi – What can you do?

- The attack targets the system and can lead to the following;
  - Circumvent access controls
  - Read sensitive data
  - Modify data (insert/update/delete)
  - Execute administration operations on the DB
  - Read the content of a file on the DBMS file system
    - e.g., `/etc/passwd`
  - Execute Operating System Commands

SQLi – What can you do?

[redacted]

SQLis – famous hacks

Where	Date	What Happened	Source
Fortnite	2019-01	Hackers could take control over any Fortnite player's account	Fortnite flaws allowed hackers to takeover gamers' accounts
Cisco	2018-10	The vulnerability allowed attackers to gain shell access to systems on which the license manager was deployed.	Cisco Prime License Manager SQL Injection Vulnerability
VTech	2015-11	Information on 4.8 million people who bought kids toys. Names, email, home address, passwords, etc.	One of the Largest Hacks Yet Exposes Data on Hundreds of Thousands of Kids
Tesla	2014-02	Access to all user information	Tesla Motors blind SQL Injection
LinkedIn	2012-06	6.5 million hashed passwords	LinkedIn Confirms Account Passwords Hacked

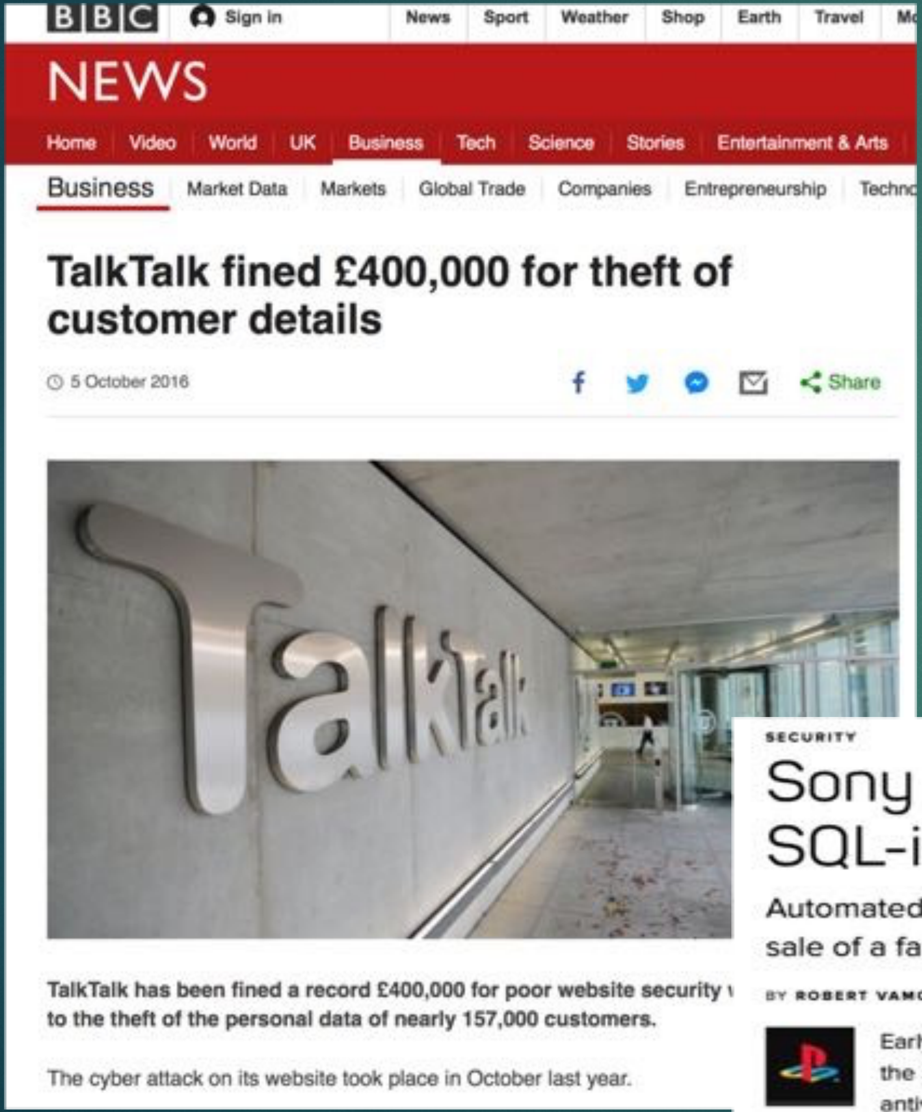
Latest SQLi News: https://portswigger.net/daily-swig/sql-injection

Aftermath of an SQLi Breach

- A 17-year-old boy has admitted hacking offences linked to a data breach at the communications firm TalkTalk, claiming he was "just showing off" to friends.
- He found the SQLi vulnerability in TALKTALK by accident, after learning from a friend how to use SQLMap
- After giving him a 12-month rehabilitation order, chairman of the bench Jean Bonnick told him: "Your IT skills will always be there - just use them legally in the future."
- The firm said the fallout from the cyber attack had cost it £42m.

http://www.bbc.co.uk/news/uk-37990246
http://uk.businessinsider.com/talktalk-hack-vamp-c-glubz-hackers-interviews-2015-11
http://krebsonsecurity.com/2015/11/talktalk-script-kids-the-quest-for-og/

SQLi in the news

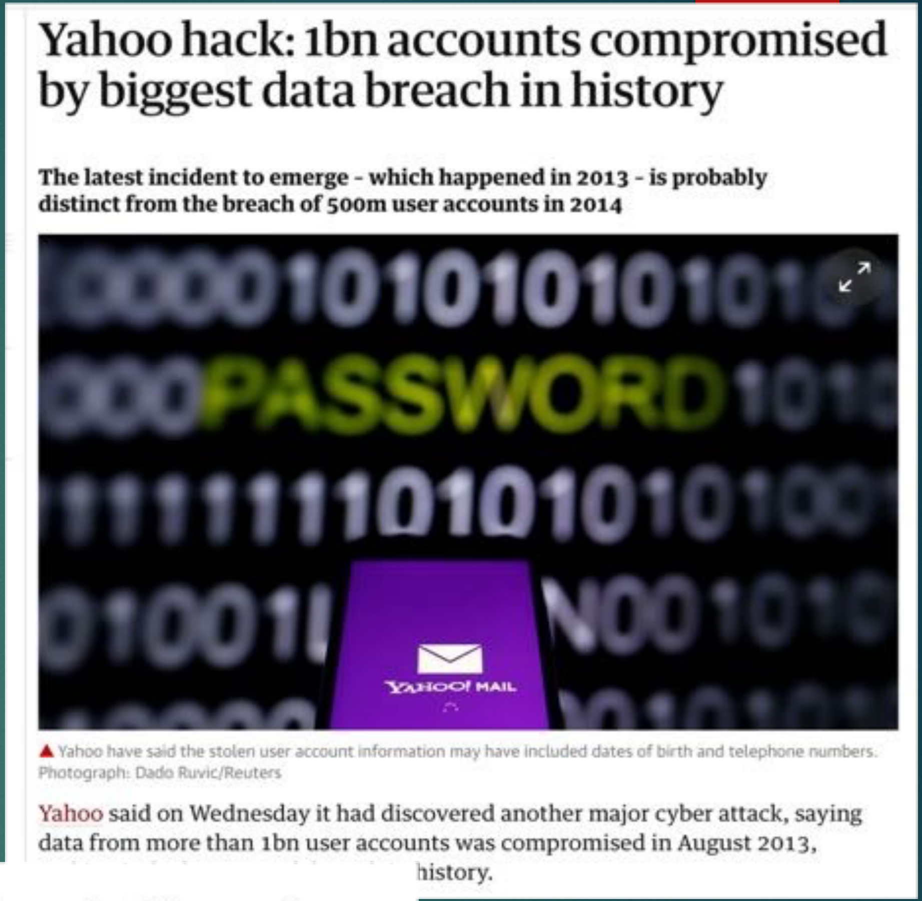


TalkTalk fined £400,000 for theft of customer details

5 October 2016

TalkTalk has been fined a record £400,000 for poor website security
to the theft of the personal data of nearly 157,000 customers.

The cyber attack on its website took place in October last year.

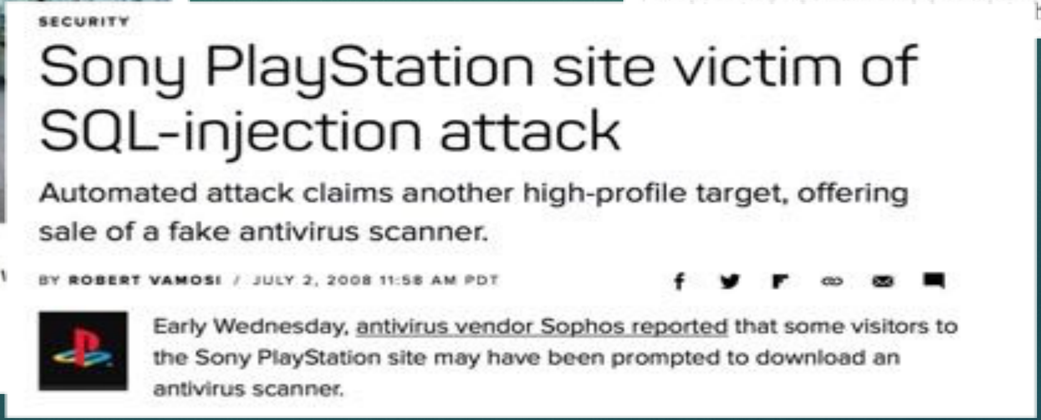


Yahoo hack: 1bn accounts compromised by biggest data breach in history

The latest incident to emerge – which happened in 2013 – is probably distinct from the breach of 500m user accounts in 2014

Yahoo have said the stolen user account information may have included dates of birth and telephone numbers. Photograph: Dado Ruvic/Reuters

Yahoo said on Wednesday it had discovered another major cyber attack, saying data from more than 1bn user accounts was compromised in August 2013,
history.



SECURITY

Sony PlayStation site victim of SQL-injection attack

Automated attack claims another high-profile target, offering sale of a fake antivirus scanner.

BY ROBERT VAMOSI / JULY 2, 2008 11:58 AM PDT

Early Wednesday, antivirus vendor Sophos reported that some visitors to the Sony PlayStation site may have been prompted to download an antivirus scanner.

Detection Techniques

- Find out **where the application interacts with the backend DB**. Typically the following functionality will;
  - Authentication forms
  - Search Functions
  - E-Commerce Functions, e.g., price, description, availability

- Make a list of all input fields
  - including the hidden fields
  - Consider HTTP headers and Cookies

Single Quote '

- The very first test should be with;
  - a single quote ‘
  - a semicolon ;
- Single Quote in SQL is used for a string terminator
- A semicolon in SQL is used to end an SQL statement
- The addition of these in an application input can cause an error if they are not dealt with in a secure way by the application

Fingerprinting the DB

- Structured Query Language (SQL) is standard
  - However, each DBMS differs; special commands, functions to retrieve data, features etc.
- First method is to observe the error returned by the DB where each DB will throw a different error message
- Second method can be used in the case of:
  - No error message
  - Custom error message

MySql: ‘test’ + ‘ing’
SQL Server: ‘test’ ‘ing’
Oracle: ‘test’ | | ’ing’

Security Shepherd SQLi Lesson

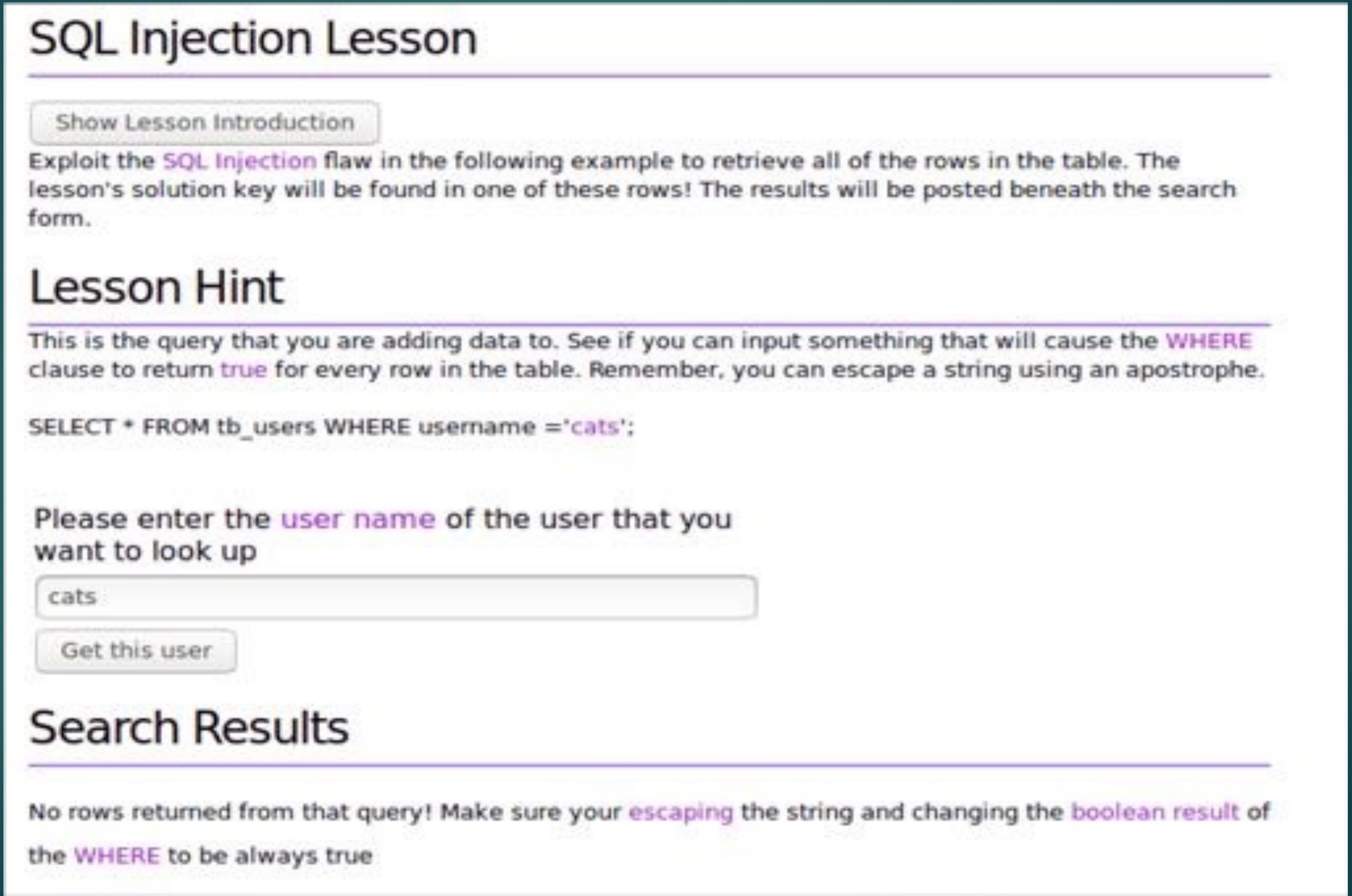
- The next few slides will demonstrate how to:
  - Detect an SQL Injection
  - Observe an error and determine the backend DBMS
  - Exploit an SQL Injection

Dangerous SQLi Code Example

```
conn = DriverManager.getConnection(connectionURL,username,password);
Statement stmt;
stmt = conn.createStatement();
ResultSet resultSet = stmt.executeQuery("SELECT * FROM tb_users WHERE
username = '" + username + "'");
```

Security Shepherd - https://github.com/OWASP/SecurityShepherd/blob/dev/SecurityShepherdCore/src/servlets/module/lesson/SqlInjectionLesson.java

Discovering SQL Injection



SQL Injection Lesson

Exploit the SQL Injection flaw in the following example to retrieve all of the rows in the table. The lesson's solution key will be found in one of these rows! The results will be posted beneath the search form.

Lesson Hint

This is the query that you are adding data to. See if you can input something that will cause the WHERE clause to return true for every row in the table. Remember, you can escape a string using an apostrophe.

```
SELECT * FROM tb_users WHERE username ='cats';
```

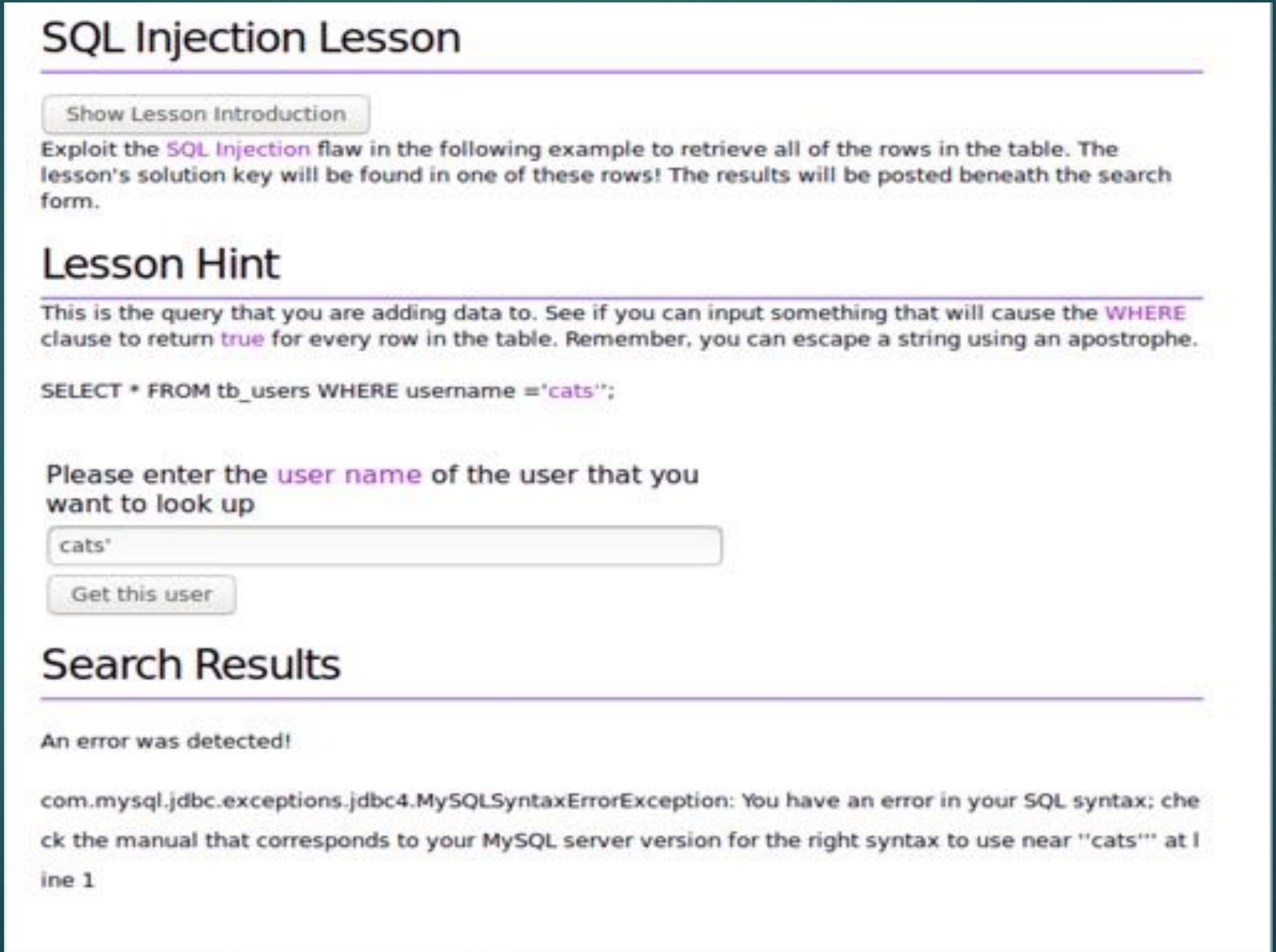
Please enter the user name of the user that you want to look up

cats

Search Results

No rows returned from that query! Make sure your escaping the string and changing the boolean result of the WHERE to be always true

Discovery - Single Quote



SQL Injection Lesson

Show Lesson Introduction

Exploit the SQL Injection flaw in the following example to retrieve all of the rows in the table. The lesson's solution key will be found in one of these rows! The results will be posted beneath the search form.

Lesson Hint

This is the query that you are adding data to. See if you can input something that will cause the WHERE clause to return true for every row in the table. Remember, you can escape a string using an apostrophe.

```
SELECT * FROM tb_users WHERE username ='cats'';
```

Please enter the user name of the user that you want to look up

cats'

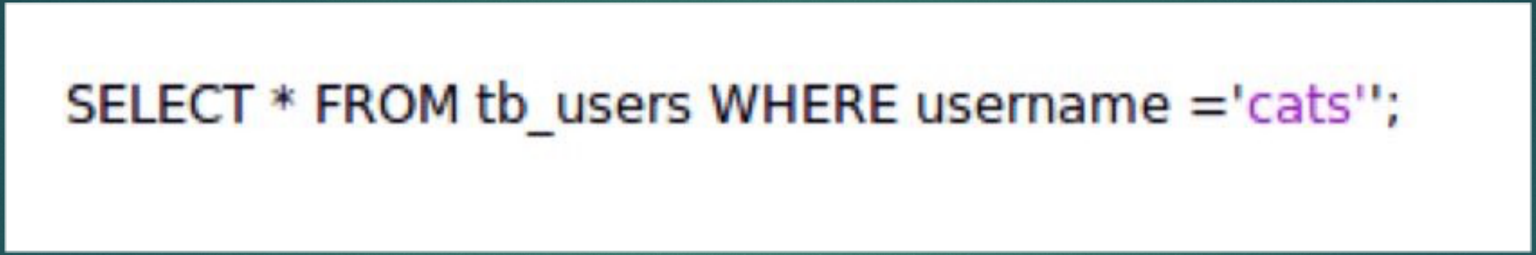
Get this user

Search Results

An error was detected!

```
com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: You have an error in your SQL syntax; che
ck the manual that corresponds to your MySQL server version for the right syntax to use near ''cats''' at l
ine 1
```

Discovery - Closer Look



```
SELECT * FROM tb_users WHERE username ='cats'';
```

SELECT * FROM tb_users

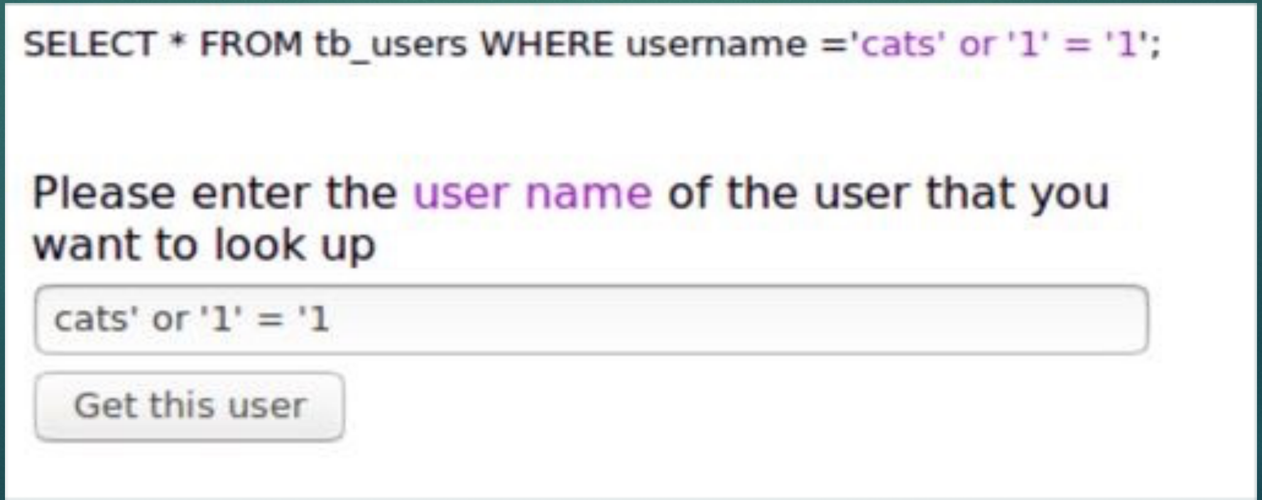
WHERE username =**'cats''**;

Exploit – Boolean True

- Query in application that is being run on the DB Server:

```
SELECT * FROM tb_users WHERE username ='cats' or '1' = '1';
```

- Attackers input string: **cats' or '1' = '1**
- *Note the application appends the first and last single quote
  - Attacker needs to enter a single quote to close the previous query (after cats)
  - Leave out a closing single quote because the application is doing this for the attacker (after the last 1)



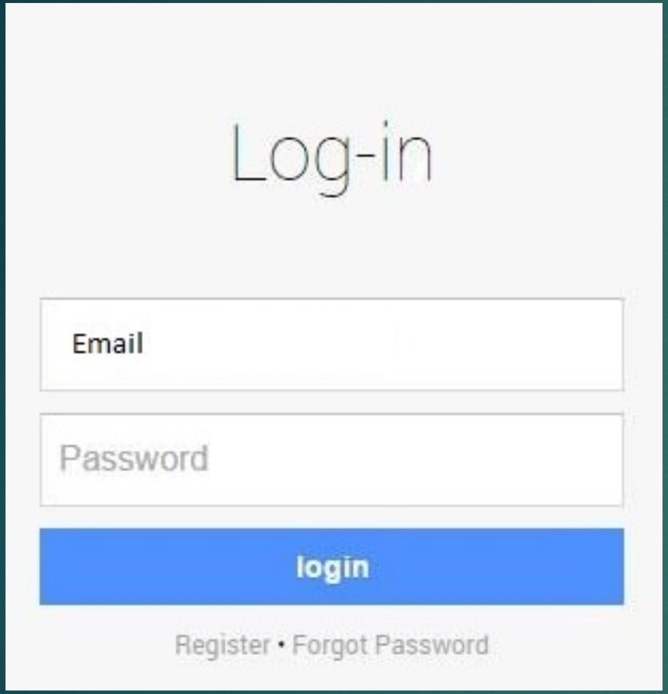
```
SELECT * FROM tb_users WHERE username ='cats' or '1' = '1';
```

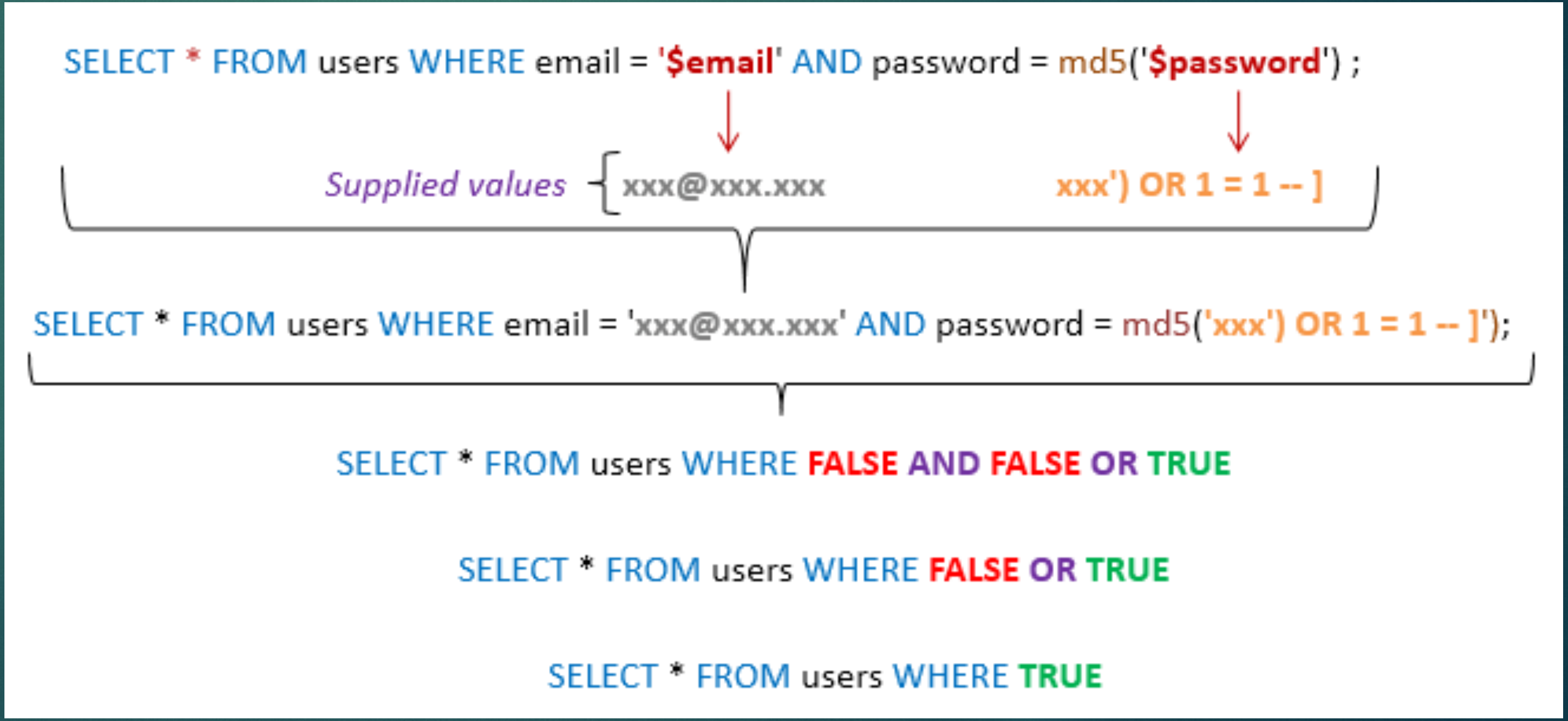
Please enter the user name of the user that you want to look up

cats' or '1' = '1

Another SQLi example

```
1:admin:zxcvbnm
2:JohnSmith:Password12345
3:BettieCharles:qwerty54
4:rondavidson:111111
5:kathywright:3rjs1la7qe
..etc
```





```
SELECT * FROM users WHERE email = '$email' AND password = md5('$password') ;
```

Supplied values { xxx@xxx.xxx   xxx') OR 1 = 1 --]

```
SELECT * FROM users WHERE email = 'xxx@xxx.xxx' AND password = md5('xxx') OR 1 = 1 -- ]');

SELECT * FROM users WHERE FALSE AND FALSE OR TRUE

SELECT * FROM users WHERE FALSE OR TRUE

SELECT * FROM users WHERE TRUE
```

SQL Injection Classes

- 3 SQL classes:
  1. **Inband**: data is extracted using the same channel that is used to inject the SQL code. This is the most straightforward kind of attack, in which the retrieved data is presented directly in the application web page.
  2. **Out-of-band**: data is retrieved using a different channel (e.g., an email with the results of the query is generated and sent to the tester).
  3. **Blind**: there is no actual transfer of data, but the tester is able to reconstruct the information by sending particular requests and observing the resulting behavior of the DB Server.
- When trying to exploit an SQL Injection there are two outcomes for the attacker:
  1. The application returns an error message generated by an incorrect query (makes it easier for the attacker to reconstruct the logic)
  2. The application hides the error details, then the tester must be able to reverse engineer the logic of the original query.

3 SQLi Techniques

1. **Union Technique**
   - can be used when the SQL injection flaw happens in a SELECT statement, making it possible to combine two queries into a single result or result set.
2. **Boolean Technique**
   - use Boolean condition(s) to verify whether certain conditions are true or false.
3. **Time delay technique**
   - use database commands (e.g. sleep) to delay answers in conditional queries. It is useful when attacker doesn’t have some kind of answer (result, output, or error) from the application.

Union Technique

- The UNION operator is used in an SQL injections attack to join a forged query with a query in the application
- This allows for the values of columns of other tables to be obtained.
- Take the following query;
  - **SELECT Name, Phone, Address FROM Users WHERE Id=$id**
- Let's UNION the query with our own query
  - **$id=1 UNION ALL SELECT creditCardNumber,1,1 FROM CreditCardTable**
- The following query is produced;
  - **SELECT Name, Phone, Address FROM Users WHERE Id=1 UNION ALL SELECT creditCardNumber,1,1 FROM CreditCardTable**
- The keyword **ALL** is necessary to get around queries that use the keyword **DISTINCT**
- An equal number of parameters/columns are needed to avoid a syntax error. Note the trailing 1s included after the creditCardNumber column

Union Technique

- **Find the right number of columns in the SELECT statement**
  - Utilize the "ORDER BY" clause followed by a number
    e.g. ORDER BY 10 --
  - If successful there are 10 columns. If not there must be fewer
- **Find out column type**
  - Use NULL value to help
  - (Assuming there are 3 columns) UNION SELECT 1,null,null--
  - If it executes the column can be replaced with an integer UNION SELECT 1,null,null--
  - Error: All cells in a column must have the same datatype – query failed
  - Try string, date, integer etc. UNION SELECT '1',1,2--

Boolean Technique

- Useful in Blind SQL Injection situations
  - No SQL error returned e.g. returns a 500 or 404 redirect or custom error message
- Inference methods:
  - Carry out Boolean queries
  - Observe the response from the server
  - Deduce the meaning of the responses

Time delay technique

- Another technique when faced with blind SQL Injection
- Send a query with a conditional statement while utilising some "sleep" function of the DBMS
- Monitor the time taken for the server to respond
  - Delay = the condition is true
  - No delay = the condition is false
- MySQL to discover the version of the DB
  - **`AND IF(version() like ‘5%’, sleep(10), ‘false’))--`**
  - If the version is 5.x and wait 10 seconds if it is

Other SQLi Techniques

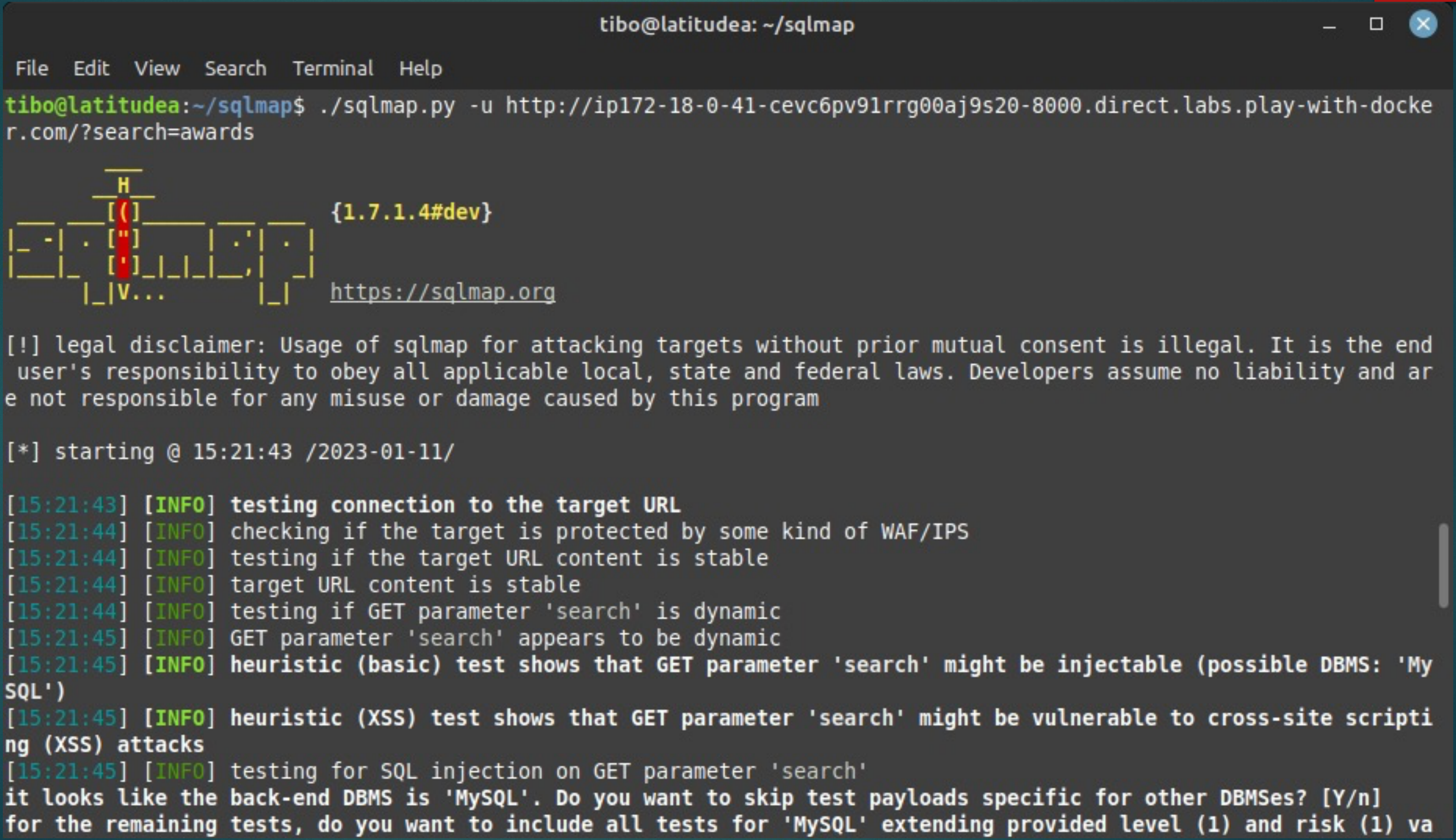
- **Error based technique**
  - When the SQLi can't be exploited to retrieve data instead Force DB to run a command when it's throwing an error
  - Exploitation is different from DB to DB
- **Out of band technique**
  - Used in blind SQL injection situations
  - Force the DB to perform an "out-of-band" connection and deliver the results of the injection to a server an attacker owns.
  - e.g., an email, a HTTP connection, etc.

WSTG-INPV-05 in the OWASP Testing Guide 4.2: https://github.com/OWASP/wstg/releases/download/v4.2/wstg-v4.2.pdf

Automated Tools

- **sqlmap**
  - De facto tool of choice. Open source tool for detecting and exploiting SQL injection vulnerabilities. Can be run as a server connecting other hacking tools to it.
- **Burp Suite Pro**
  - Active scanner will dynamically scan the application while the pentester is free to use the tool to carry out other work is paid. Comes with a plugin for sqlmap
- **IBM Security AppScan Standard**
  - Dynamic Application Scanner run at runtime against a live system
- **IBM Security AppScan Source**
  - Static Application scanner run at the build phase looking at the code line by line

SQLMap in Action



tibo@latitudea: ~/sqlmap

```
tibo@latitudea:~/sqlmap$ ./sqlmap.py -u http://ip172-18-0-41-cevc6pv91rrg00aj9s20-8000.direct.labs.play-with-docke
r.com/?search=awards
        ___
       __H__
 ___ ___[(]_____ ___ ___  {1.7.1.4#dev}
|_ -| . ["]     | .'| . |
|___|_  [']_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end
 user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and ar
e not responsible for any misuse or damage caused by this program

[*] starting @ 15:21:43 /2023-01-11/

[15:21:43] [INFO] testing connection to the target URL
[15:21:44] [INFO] checking if the target is protected by some kind of WAF/IPS
[15:21:44] [INFO] testing if the target URL content is stable
[15:21:44] [INFO] target URL content is stable
[15:21:44] [INFO] testing if GET parameter 'search' is dynamic
[15:21:45] [INFO] GET parameter 'search' appears to be dynamic
[15:21:45] [INFO] heuristic (basic) test shows that GET parameter 'search' might be injectable (possible DBMS: 'My
SQL')
[15:21:45] [INFO] heuristic (XSS) test shows that GET parameter 'search' might be vulnerable to cross-site scripti
ng (XSS) attacks
[15:21:45] [INFO] testing for SQL injection on GET parameter 'search'
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n]
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) va
```

SQL Injection Mitigation

- There are several options and each one should be considered for each vulnerability.
- Applications differ greatly and one technique may not be suitable for all
- They include:
  - **Parameterised Queries**
  - **Stored Procedures**
  - **Escaping User Supplied Input**

Parameterised Queries

- Prepared Statements or Parameterised Queries is how all developers should be thought how to write a DB call
- Simple to write and easier to understand than dynamic queries
- Define SQL code and then pass in each parameter to the query after allowing for the **designation between code and data**
    - Each language has its own function to achieve this;
        - Java EE – use PreparedStatement() with bind variables
    - .NET – use parameterized queries like SqlCommand() or OleDbCommand() with bind variables
    - PHP – use PDO with strongly typed parameterized queries (using bindParam())
    - Hibernate - use createQuery() with bind variables (called named parameters in Hibernate)
    - SQLite - use sqlite3_prepare() to create a statement object
- In rare circumstances, prepared statements can harm performance
- In conjunction always use input validation or proper escaping

Param Query:Code Example

```
String custname = request.getParameter("customerName");
// This should REALLY be validated too
// perform input validation to detect attacks
String query = "SELECT account_balance FROM user_data WHERE
user_name = ? ";

PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, custname);
ResultSet results = pstmt.executeQuery( );
```

Stored Procedures

- Stored procedures are not always safe from SQL injection!
- SQL code for a stored procedure is defined and stored in the database itself and then called from the application
- Being safe with this means not including any dynamic SQL generation
- In conjunction always use input validation or proper escaping
- Verify the user role that the stored procedures are executing under (this could lead to an attacker gaining full access to the DB)

Stored Procedure: Code Example

```
String custname = request.getParameter("customerName");
// This should REALLY be validated

try {
   CallableStatement cs = connection.prepareCall("{call
sp_getAccountBalance(?)}");
   cs.setString(1, custname);
   ResultSet results = cs.executeQuery();
   // … result set handling
} catch (SQLException se) {
   // … logging and error handling
}
```

Other SQLi defences

- **Whitelist input validation**
  - User input is mapped to legal table names and columns
  - For sorting order convert user input to boolean
- **Escaping User Supplied Input**
  - Last resort defense if others are not feasible.
  - Difficult to get right and database specific
  - Used on legacy applications new applications should use parameterised statements or stored procedures
  - Each DBMS supports one or more character escparing schemes
  - If all user supplied input is escaped properly the DBMS will not confuse input with SQL code
- **Avoid detailed error messages**
- **Enforce least privilege**