Securing Web Applications with FFP (FAUST, flagbot, polygl0ts)

SQL Injection

Florian Hantke [1]

In collaboration with [1]FAUST (ctf@fsi.informatik.uni-erlangen.de), [2]flagbot (ctf@vis.ethz.ch) and [3]polygl0ts (ctf@epfl.ch)

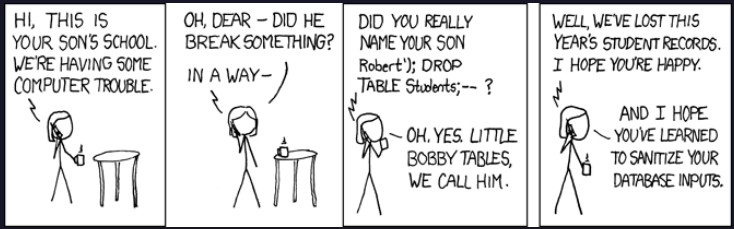
October 9, 2021

Table of Contents

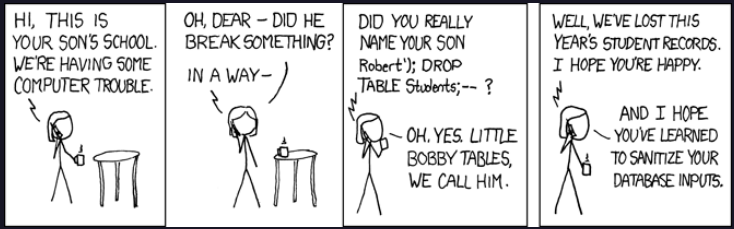
Introduction



What's your level of experience?

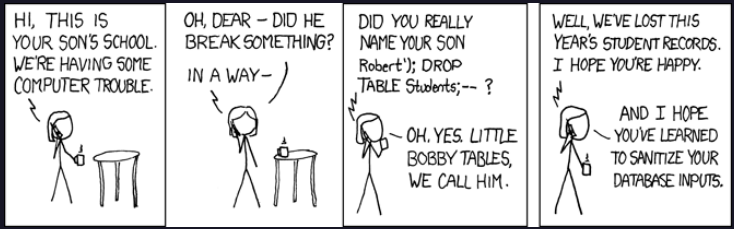


What's your level of experience?



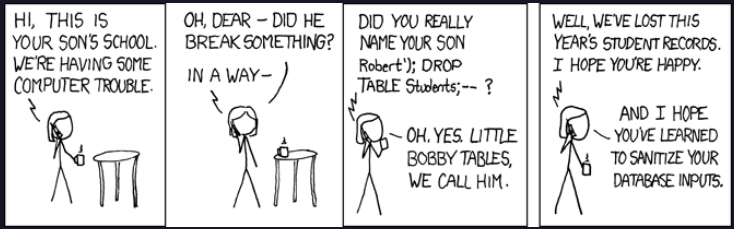
- Do you know what SQL is?

What's your level of experience?



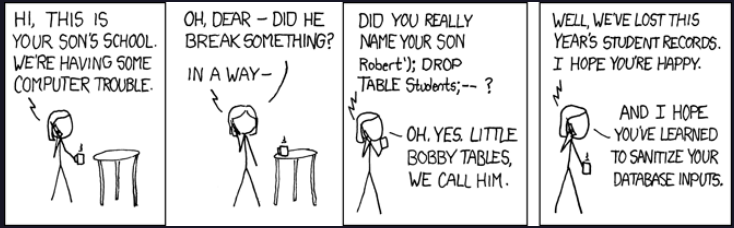
- Do you know what SQL is?
- Have you ever used SQL?

What's your level of experience?



- Do you know what SQL is?
- Have you ever used SQL?
- Do you know what a SQL Injection (SQLi) is?

What's your level of experience?

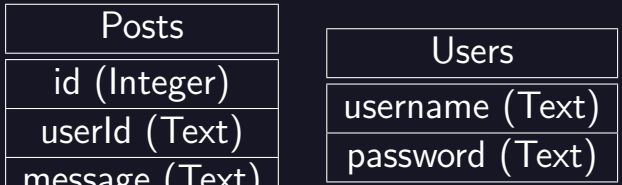


- Do you know what SQL is?
- Have you ever used SQL?
- Do you know what a SQL Injection (SQLi) is?
- Have you ever performed a SQLi?

The basics of SQL



Our database



Our database

id	userId	message
1	Alice	Hello World
2	Bob	This is my message
3	Florian	Keep on hacking

username	password
Admin	1a23f164c598...6cd472fe289
Alice	ed2542a27491...56cee399238
Bob	b5320173a05a...a513584274f
Florian	eaf47af556eb...454c3168025

Select

Select something from a database:

```
SELECT
  id, userId, message
FROM posts
WHERE userId = 'Florian'
```

- id, userId, mesage are the fields you select
- FROM defines the table that you read
- WHERE is the condition the result must fulfill
- Result: (3, 'Florian', 'Keep on hacking')



Insert

Insert something into a database:

```
INSERT INTO posts
VALUES (4, 'Alice', 'This is my first message')

INSERT INTO posts (userId, message)
VALUES ('Florian', 'This is already my second post')
```

- INSERT INTO <table name>(<optional columns>)
- VALUES (<values>)
- Insert two posts. One post with id=4 for user Alice and one post without id for user Florian.

Update

Update one or more entries in a database:

```
UPDATE posts SET userId='Florian'
WHERE id = 4
```

- UPDATE <table name>SET <field>=<value>)
- WHERE <condition>
- Change the user of the post with id=4 to Florian

Delete

Delete one or more entries in a database:

```
DELETE FROM posts WHERE id = 4
DELETE FROM posts WHERE message LIKE '%already%'
```

- DELETE FROM <table>WHERE <condition>
- Delete all posts with id=4
- Delete all posts with a message including 'already'

Join

Join one or more tables together.

```
SELECT message, password FROM posts JOIN users ON (userId=username)
```

- SELECT <fields>FROM <table 1>JOIN <table 2>ON (<condition>)

message	password
Hello World	ed2542a27491...56cee399238
This is my message	b5320173a05a...a513584274f
...	...

SQL Injection

SQL Injection

SQL injection is an attack in which malicious code is inserted into strings that are later passed to an instance of SQL Server for parsing and execution. (Microsoft)

SQL Injection

Never copy this login logic:

```
$qry = "SELECT * FROM users WHERE username='".$_POST["u"]
        ."'AND password='".$_POST["p"]."'";
$res = mysql_query($qry);
$row = mysql_fetch_array($res);
if ($row) {
    login($_POST["u"])
}
```

Example: Authentication Bypass

Login query:

```
SELECT * FROM users
WHERE username = '<userinput>' AND password = '<hash_of_userinput>'
```

Example: Authentication Bypass

Login query:

```
SELECT * FROM users
WHERE username = '<userinput>' AND password = '<hash_of_userinput>'
```

Normal use, foo / bar:

```
SELECT * FROM users
WHERE username = 'foo' AND password = 'fcde2b2...'
```

Example: Authentication Bypass

Login query:

```
SELECT * FROM users
WHERE username = '<userinput>' AND password = '<hash_of_userinput>'
```

Normal use, foo / bar:

```
SELECT * FROM users
WHERE username = 'foo' AND password = 'fcde2b2...'
```

Malicious use, Admin' -- / whatever:

```
SELECT * FROM users
WHERE username = 'Admin' -- ' AND password = '1a23f16...'
```

Example: Union Select

Posts query:

```
SELECT * FROM posts WHERE message LIKE '%<userinput>%'
```

Example: Union Select

Posts query:

```
SELECT * FROM posts WHERE message LIKE '%<userinput>%'
```

Normal use, searching for Hello:

```
SELECT * FROM posts WHERE message LIKE '%Hello%'
```

Example: Union Select

Posts query:

```
SELECT * FROM posts WHERE message LIKE '%<userinput>%'
```

Normal use, searching for Hello:

```
SELECT * FROM posts WHERE message LIKE '%Hello%'
```

Malicious use, searching for ' union select 1, username, password from users #:

```
SELECT * FROM posts WHERE message LIKE '%' union select
1, username, password from users #%'
```



Example: Union Select

id	userId	message
1	Alice	Hello World
1	Admin	1a23f164c598...6cd472fe289
1	Alice	ed2542a27491...56cee399238
1	Bob	b5320173a05a...a513584274f
1	Florian	eaf47af556eb...454c3168025

Useful SQL commands

- End a query: ';'

Useful SQL commands

- End a query: ';'
- Comments: --, /*, #

Useful SQL commands

- End a query: ';'
- Comments: --, /*, #
- LIMIT x,y (MySQL): output from line x a total of y lines

Useful SQL commands

- End a query: ';'
- Comments: --, /*, #
- LIMIT x,y (MySQL): output from line x a total of y lines
- ORDER BY: Order of rows

Useful SQL commands

- End a query: ';'
- Comments: --, /*, #
- LIMIT x,y (MySQL): output from line x a total of y lines
- ORDER BY: Order of rows
- OR 1=1: The WHERE condition is always true

Useful SQL commands

- End a query: ';'
- Comments: --, /*, #
- LIMIT x,y (MySQL): output from line x a total of y lines
- ORDER BY: Order of rows
- OR 1=1: The WHERE condition is always true
- HAVING: New WHERE condition if no GROUP BY

Useful SQL commands

- End a query: ';'
- Comments: --, /*, #
- LIMIT x,y (MySQL): output from line x a total of y lines
- ORDER BY: Order of rows
- OR 1=1: The WHERE condition is always true
- HAVING: New WHERE condition if no GROUP BY
- LOAD_FILE() / SELECT ... INTO FILE: Read / Write files

Useful SQL commands

- End a query: ';'
- Comments: --, /*, #
- LIMIT x,y (MySQL): output from line x a total of y lines
- ORDER BY: Order of rows
- OR 1=1: The WHERE condition is always true
- HAVING: New WHERE condition if no GROUP BY
- LOAD_FILE() / SELECT ... INTO FILE: Read / Write files
- Tools: Sqlmap, Python, your brain

MySQL



MySQL

- Database **mysql**: Contains all users and its privileges. That can be very interesting for attacks, i.e. mysql.user

MySQL

- Database **mysql**: Contains all users and its privileges. That can be very interesting for attacks, i.e. mysql.user
- Database **information_schema**: Contains all information about the databases stored on the server. This includes tables and structures.

MySQL

- Database **mysql**: Contains all users and its privileges. That can be very interesting for attacks, i.e. mysql.user
- Database **information_schema**: Contains all information about the databases stored on the server. This includes tables and structures.
- Read the documentation or SQLi cheat sheets for more tricks :)

Mitigation Techniques



Mitigation Techniques

Mitigation Techniques

- Use prepared statements

```
# Never do this:
cur_db.execute("SELECT * FROM users WHERE username = '%s'" % username)
# Do this:
cur_db.execute("SELECT * FROM users WHERE username = '?'", username)
```

Mitigation Techniques

- Use prepared statements

```
# Never do this:
cur_db.execute("SELECT * FROM users WHERE username = '%s'" % username)
# Do this:
cur_db.execute("SELECT * FROM users WHERE username = '?'", username)
```

- Input validation

Mitigation Techniques

- Use prepared statements

```
# Never do this:
cur_db.execute("SELECT * FROM users WHERE username = '%s'" % username)
# Do this:
cur_db.execute("SELECT * FROM users WHERE username = '?'", username)
```

- Input validation
- Escaping all user-supplied input

Mitigation Techniques

- Use prepared statements

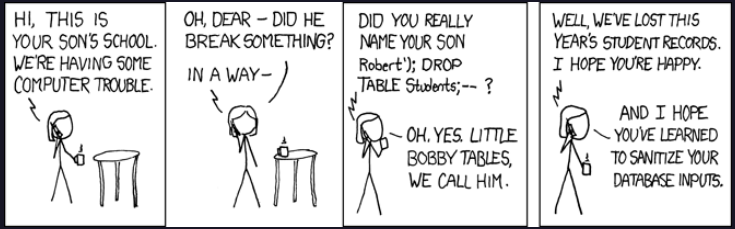
```
# Never do this:
cur_db.execute("SELECT * FROM users WHERE username = '%s'" % username)
# Do this:
cur_db.execute("SELECT * FROM users WHERE username = '?'", username)
```

- Input validation
- Escaping all user-supplied input
- Multiple database users with specific previleges. Never use the root user with your application.

Useful Links



Useful Links



- https://book.hacktricks.xyz/pentesting-web/sql-injection
- https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
- https://owasp.org/www-community/attacks/SQL_Injection