CS 88: Security and Privacy

08: Web Security + SQL Injection!

09-22-2022

slides adapted from Dave Levine, Deian Stefan, Vitaly Shmatikov, Wenliang Du
A very basic web architecture

DB is a separate entity, logically (and often physically)
Databases management systems: DBMS

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
<th>Email</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpongeBob</td>
<td>20</td>
<td><a href="mailto:sponge@ocean.com">sponge@ocean.com</a></td>
<td>12345!!</td>
</tr>
<tr>
<td>Squidward</td>
<td>60</td>
<td><a href="mailto:squiddy@ocean.com">squiddy@ocean.com</a></td>
<td>clarinet%%</td>
</tr>
<tr>
<td>Patrick Star</td>
<td>21</td>
<td><a href="mailto:patrick@ocean.com">patrick@ocean.com</a></td>
<td>theStar5</td>
</tr>
<tr>
<td>Mr. Krabs</td>
<td>55</td>
<td><a href="mailto:krusty@ocean.com">krusty@ocean.com</a></td>
<td>noFreelunch</td>
</tr>
</tbody>
</table>

- Database provides data storage and manipulation
- Programmers query the database

Database Management Systems Provide:
- **semantics** for organizing data
- a **language** for querying data
- **APIs** for interoperability (w/other languages)
- **management**: via users + permissions
Databases: basics

<table>
<thead>
<tr>
<th>Users</th>
<th>Table Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Age</td>
</tr>
<tr>
<td>SpongeBob</td>
<td>20</td>
</tr>
<tr>
<td>Squidward</td>
<td>60</td>
</tr>
<tr>
<td>Patrick Star</td>
<td>21</td>
</tr>
<tr>
<td>Mr. Krabs</td>
<td>55</td>
</tr>
</tbody>
</table>

Table

- **Name**: SpongeBob, Squidward, Patrick Star, Mr. Krabs
- **Age**: 20, 60, 21, 55
- **Email**: sponge@ocean.com, squiddy@ocean.com, patrick@ocean.com, krusty@ocean.com
- **Password**: 12345!! (SpongeBob), clarinet%% (Squidward), theStar5 (Patrick Star), noFreelunch (Mr. Krabs)
### SQL: Standard Query Language

#### Users

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
<th>Email</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpongeBob</td>
<td>20</td>
<td><a href="mailto:sponge@ocean.com">sponge@ocean.com</a></td>
<td>12345!!</td>
</tr>
<tr>
<td>Squidward</td>
<td>60</td>
<td><a href="mailto:squiddy@ocean.com">squiddy@ocean.com</a></td>
<td>clarinet%%</td>
</tr>
<tr>
<td>Patrick Star</td>
<td>21</td>
<td><a href="mailto:patrick@ocean.com">patrick@ocean.com</a></td>
<td>theStar5</td>
</tr>
<tr>
<td>Mr. Krabs</td>
<td>55</td>
<td><a href="mailto:krusty@ocean.com">krusty@ocean.com</a></td>
<td>noFreelunch</td>
</tr>
</tbody>
</table>

**SELECT** Age **FROM** Users **WHERE** Name='SpongeBob'; **Answer** = 20
SELECT Age FROM Users WHERE Name='SpongeBob'; Answer = 20

UPDATE Users SET email='patrickStar@ocean.com' WHERE Age=21; -- this is a comment

INSERT INTO Users Values ('Gary', 6, 'gary@ocean.com', 'snailmail');

DROP TABLE Users;
Server-side code

Example #1 eval() example - simple text merge

```php
<?php
  $string = 'cup';
  $name = 'coffee';
  $str = 'This is a $string with my $name in it.';
  echo $str. "\n";
  eval("\$str = \"$str\"; ");
  echo $str. "\n";
?>
```

The above example will output:

```
This is a $string with my $name in it.
This is a cup with my coffee in it.
```
Server-side code

Your program manipulates data
Data manipulates your program

Description

eval(string $code): mixed

Evaluates the given code as PHP.

Caution  The eval() language construct is very dangerous because it allows execution of arbitrary PHP code. Its use thus is discouraged. If you have carefully verified that there is no other option than to use this construct, pay special attention not to pass any user provided data into it without properly validating it beforehand.

Server-side code

Login code: (php)

```php
$result = mysql_query("SELECT * FROM Users
    WHERE (name='{$user}' and password='{$pass}');");
```

```sql
SELECT * FROM Users WHERE Name='SpongeBob'; AND password = '12345!!';
```

How can we exploit this code?
SQL Injection

Username: [input], Password: [input]

Log me on automatically each visit [ ]

Log in

```
$sresult = mysql_query("select * from Users
                                              where(name='\$user' and password='\$pass');");
```

```
$result = mysql_query("select * from Users
                                              where(name= 'spongebob' or 1=1);#
                                              and password='whocares');");
```
SQL Injection

$sresult = mysql_query("select * from Users
where(name='spongebob' and password='$pass');");

$result = mysql_query("select * from Users
where(name='spongebob' or 1=1);#
DROP TABLE Users; --
' and password='whocares');");

Can chain together statements with semicolon:
STATEMENT 1 ; STATEMENT 2
Exploits of a Mom

Hi, this is your son's school. We're having some computer trouble.

Oh, dear - did he break something?

In a way -
Exploits of a Mom

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.
Find health coverage that works for you

Get quality coverage at a price you can afford. Open enrollment in the Health Insurance Marketplace continues until March 31, 2014.

APPLY ONLINE  APPLY BY PHONE
SQL Injection: The underlying issue

```php
$result = mysql_query("select * from Users
where(name='$user' and password='$pass');");
```

- This one string combines the code and the data
- Similar to buffer overflows:

When the boundary between code and data blurs, we open ourselves up to vulnerabilities
SQL Injection: Counter measures

• Blocklists: delete characters you don’t want
  • [' ] [--] [;]

• Safelists:
  • Check that the user-provided input is in some set of values known to be safe.
    • e.g. integer within the right range
  • Given an invalid input:
    • better to reject than fix
    • “fixes” introduce new vulnerabilities
    • principle of fail-safe defaults

• Escape characters:
  • ‘ = \'
  • ; = "; ... so on
The underlying issue

```php
$result = mysql_query("select * from Users
    where(name='".$user.' and password='".$pass.'");");
```
The underlying issue

```php
$result = mysql_query("select * from Users
where(name='$user' and password='$pass');");
```
Attacks Change Query Structure

Boyd et. al [BK 04], ANCS; Buehrer et. al. [BWS 05], SEM;
Halfond et. al. [HO 05], ASE; Nguyen-Tuong et. al. [NGGSE 05], SEC; Pietraszek et. al. [PB 05], RAID; Valeur et. al. [VMV 05], DIMVA;
Su et. al. [SW 06], POPL ...

Benign Query

```
WHERE username = 'Spongebob' AND password = '12345!!'
```

Attack Query

```
WHERE username = 'Spongebob' OR 1=1# AND ...
```
SQL injection countermeasures

Prepared statements & Bind variables

Key idea: Decouple the code and the data

```
$result = mysql_query("select * from Users
where(name='$user' and password='$pass');");
```
SQL injection countermeasures

Prepared statements & Bind variables

Key idea: *Decouple* the code and the data

```php
$result = mysql_query("select * from Users
    where(name='\$user' and password='\$pass');");
```

```php
$db = new mysql("localhost", "user", "pass", "DB");

$statement = $db->prepare("select * from Users
    where(name=? and password=?);");  // Bind Variables

$statement->bind_param("ss", $user, $pass);
$statement->execute();  // Bind Variables are typed
```
SQL injection countermeasures

Prepared statements & Bind variables

Key idea: *Decouple* the code and the data

```
$result = mysql_query("select * from Users
    where(name='".$user." and password='".$pass.");");
```

```
$db = new mysql("localhost", "user", "pass", "DB");

$statement = $db->prepare("select * from Users
    where(name=? and password=?);");  

Bind Variables

Decoupling let’s us compile now, before binding the data

$statement->bind_param("ss", $user, $pass);
$statement->execute();

Bind Variables are typed
The underlying issue

```
$statement = $db->prepare("select * from Users
  where(name=? and password=?);");
```
The underlying issue

$statement = $db->\texttt{prepare("select * from Users where(name=? and password=?);")};

Prepare is only applied to the leaves, so the structure of the tree is fixed.
The underlying issue

```php
(statement = $db->prepare("select * from Users where(name=? and password=?);"));
```

Prepare is only applied to the leaves, so the structure of the tree is fixed.
Mitigating the impact

• Limit privileges
  • limit commands and/or tables a user can access
  • E.g.: Allow SELECT queries on Orders_Table but not on Creditcards_Table

• Follow the principle of least privilege

• Encrypt sensitive data