

CS 88: Security and Privacy

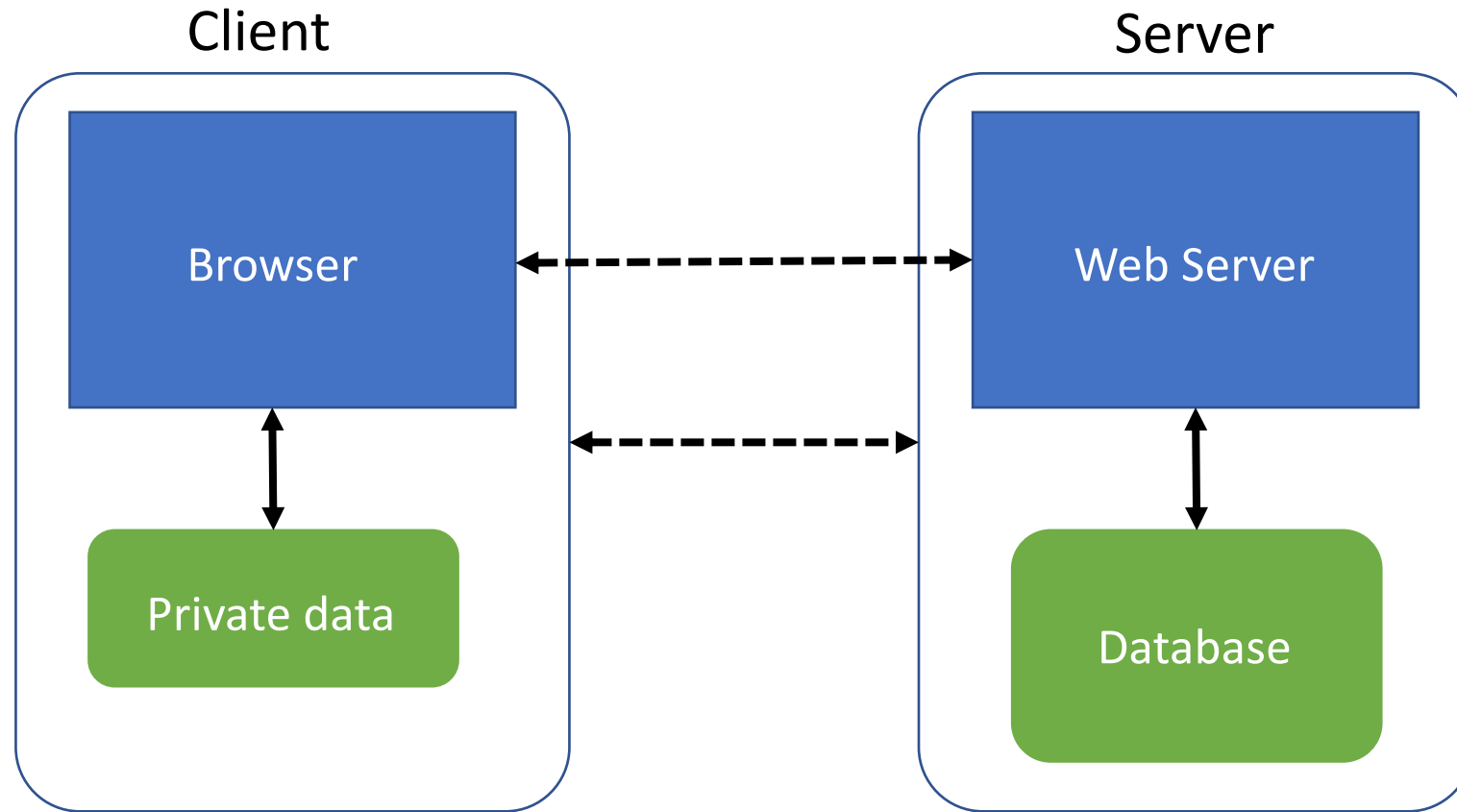
08: Web Security + SQL Injection!

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

Users

Name	Age	Email	Password
SpongeBob	20	.sponge@ocean.com	12345!!
Squidward	60	squiddy@ocean.com	clarinet%%
Patrick Star	21	patrick@ocean.com	theStar5
Mr. Krabs	55	krusty@ocean.com	noFreelunch

- 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

Users ← Table Name

Name	Age	Email	Password
SpongeBob	20	sponge@ocean.com	12345!!
Squidward	60	squiddy@ocean.com	clarinet%%
Patrick Star	21	patrick@ocean.com	theStar5
Mr. Krabs	55	krusty@ocean.com	noFreelunch

↑ Column

Row (Record)

SQL: Standard Query Language

Users

Name	Age	Email	Password
SpongeBob	20	.sponge@ocean.com	12345!!
Squidward	60	squiddy@ocean.com	clarinet%%
Patrick Star	21	patrick@ocean.com	theStar5
Mr. Krabs	55	krusty@ocean.com	noFreelunch

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

SHOW DATABASES;

```
+-----+
| Database          |
+-----+
| information_schema |
| mysql             |
| performance_schema |
| sqllab_users      |
| sys               |
+-----+
5 rows in set (0.00 sec)
```

SQL: Standard Query Language

Users

Name	Age	Email	Password
SpongeBob	20	.sponge@ocean.com	12345!!
Squidward	60	squiddy@ocean.com	clarinet%%
Patrick Star	21	patrickstar@ocean.com	theStar5
Mr. Krabs	55	krusty@ocean.com	noFreelunch
Gary	6	gary@ocean.com	snailmail



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
$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

Description

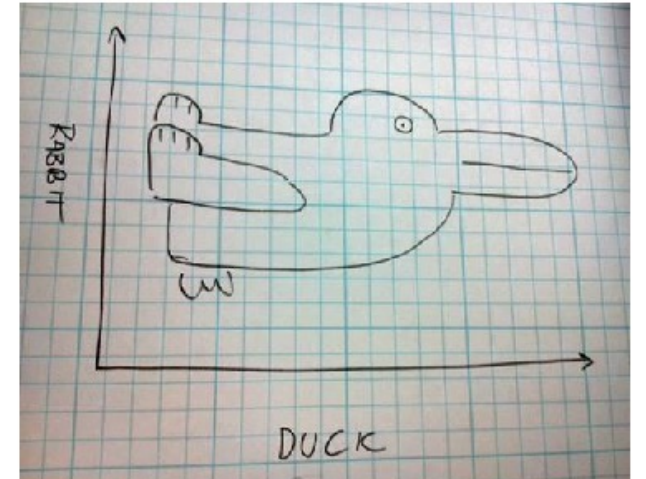
```
eval(string $code): mixed
```

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.

Your program manipulates data

Data manipulates your program



Server-side code



Username: Password: Log me on automatically each visit


Login code: (php)

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

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

How can we exploit this code?

SQL Injection



A screenshot of a web application's login interface. It features a light gray header bar containing a 'Username:' label, an empty text input field, a 'Password:' label, another empty text input field, a checkbox labeled 'Log me on automatically each visit', and a 'Log in' button. A dotted line connects the bottom of the username input field to a separate box below.

spongebob' or 1=1);#

```
$result = 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



A screenshot of a web application's login interface. It features a light gray header bar containing a 'Username:' label, an empty text input field, a 'Password:' label, another empty text input field, a checkbox labeled 'Log me on automatically each visit', and a 'Log in' button. A dotted line originates from the username input field and points to a separate box below.

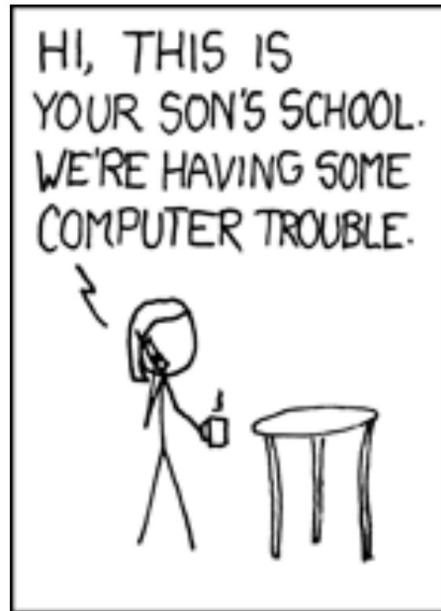
```
spongebob' or 1=1); DROP TABLE Users; #
```

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

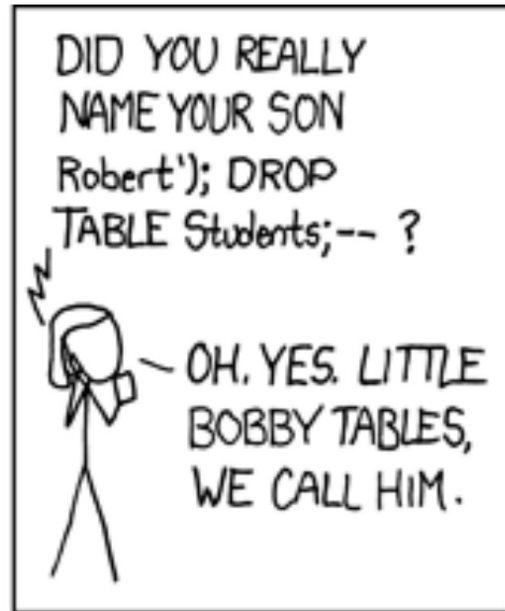
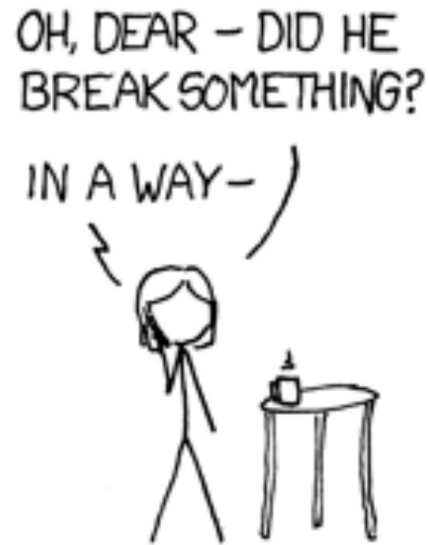
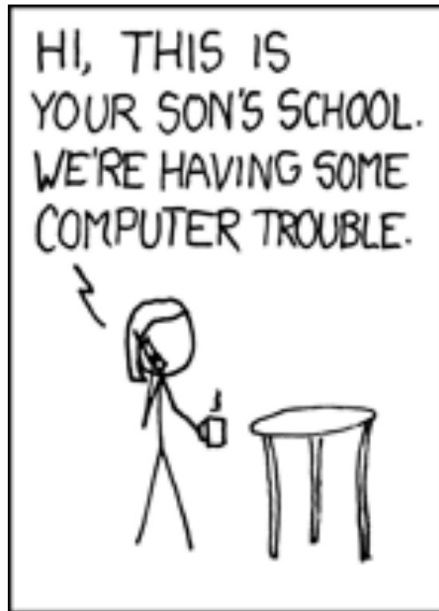
```
$result = mysql_query("select * from Users  
where(name='spongebob' or 1=1);#  
DROP TABLE Users; --  
' and password='whocares');");
```

**Can chain together statements with semicolon:
STATEMENT 1 ; STATEMENT 2**

Exploits of a Mom



Exploits of a Mom



SEARCH

Improving
HealthCare.gov

The Health Insurance Marketplace online application isn't available from a... we make improvements. Additional down times may be possible as we wo... and the Marketplace call center remain available during these hours.

```

;select * from users
;show tables;
;show tables; --
;premium payments
;select * from *;
; grant
; rehabilitative and habilitative
; show tables

```

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

4 Ways to Get Marketplace Coverage



SEE PLANS AND PRICES IN YOUR AREA

SEE PLANS NOW

Get covered: A one-

Find out if you

See 4 ways you can

Get in-person help in

Call 1-800-318-2596

SQL Injection: The underlying issue

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

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

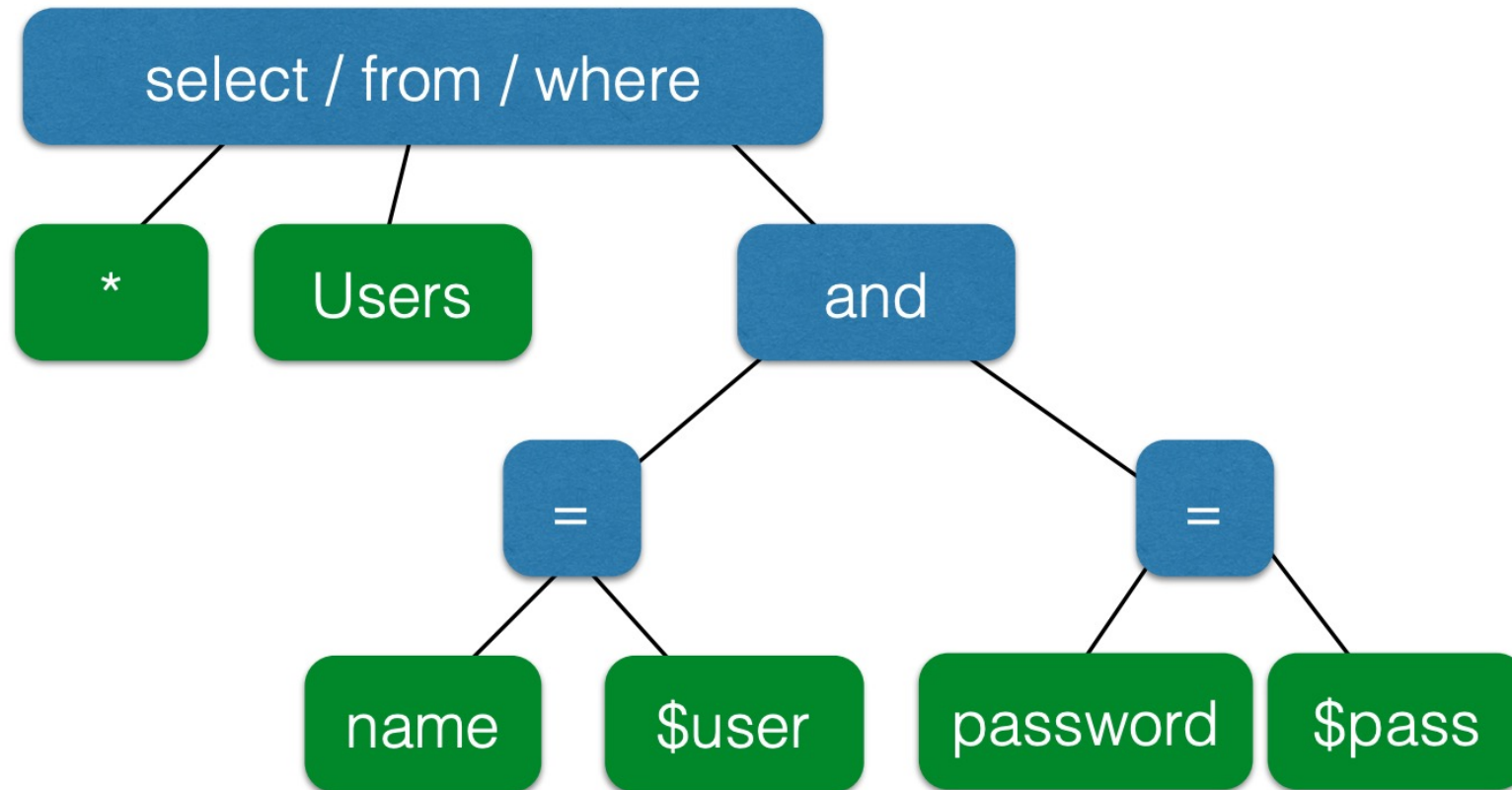
**When the boundary between code and data blurs,
we open ourselves up to vulnerabilities**

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

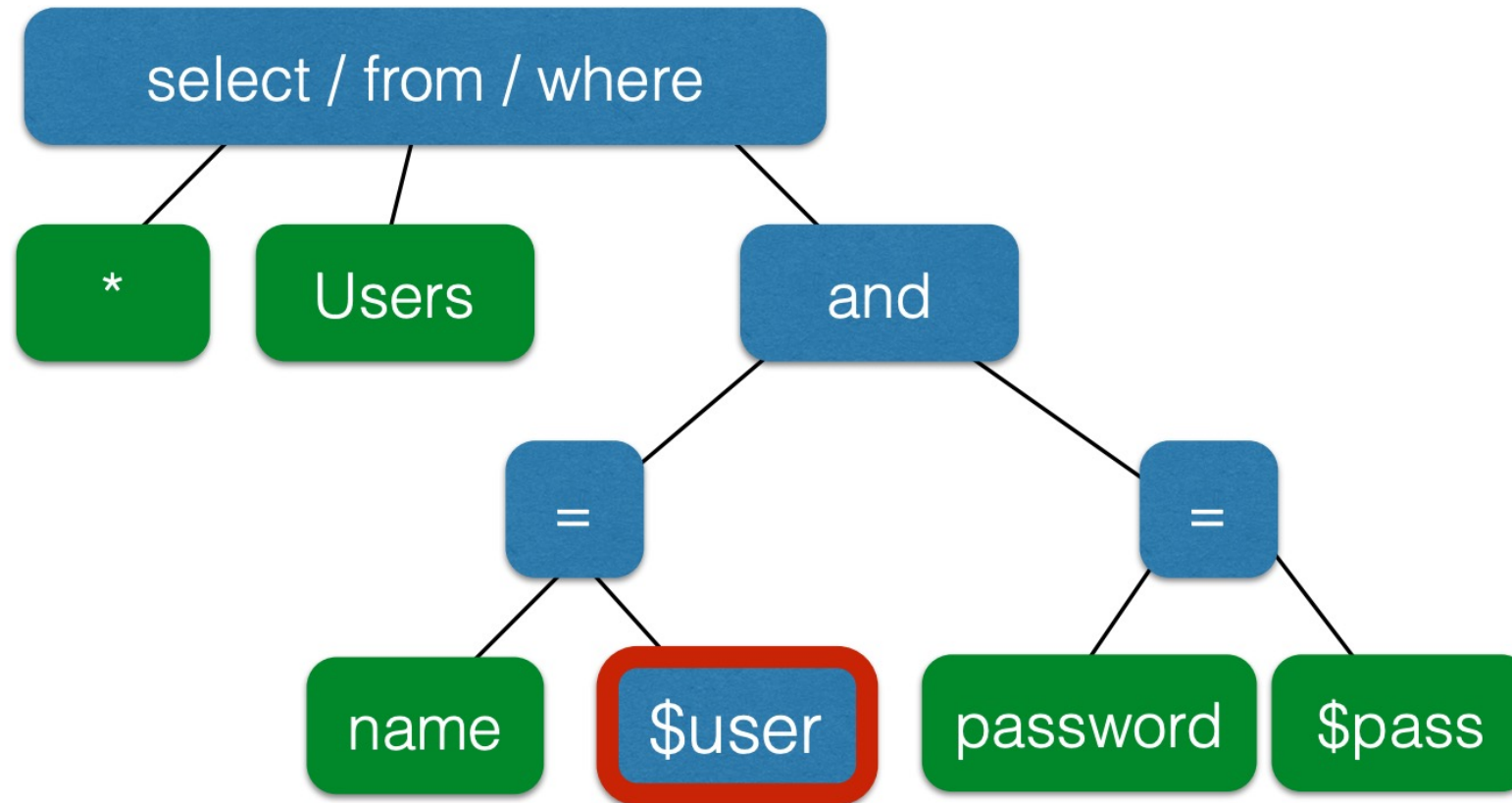
The underlying issue

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



The underlying issue

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

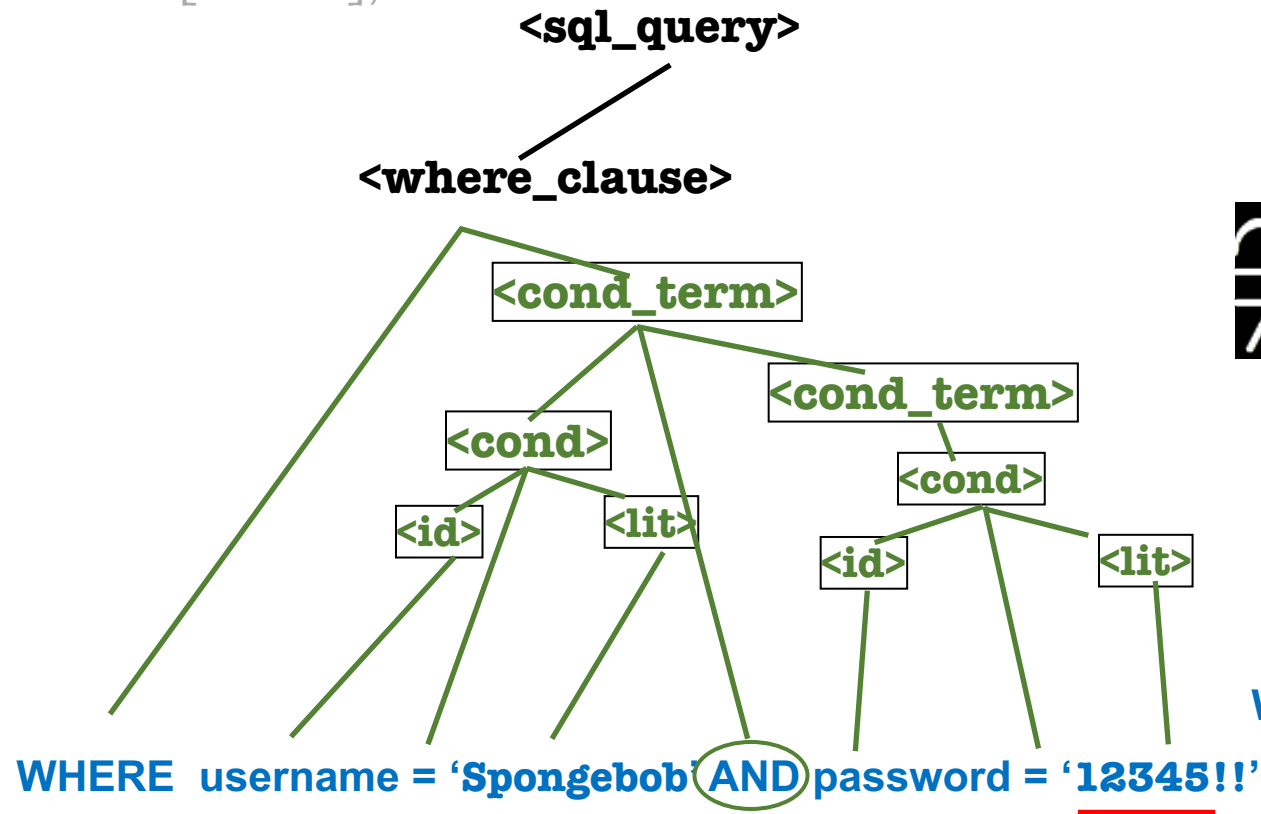


Attacks Change Query Structure

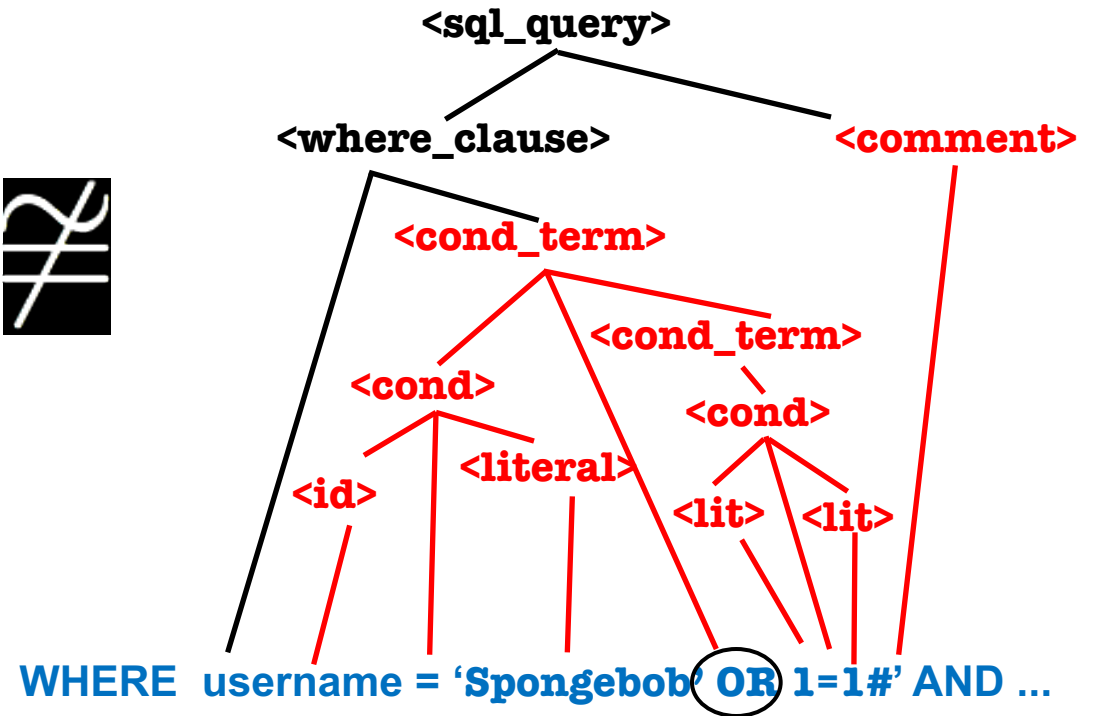
Boyd et. al [BK 04], ANCS ; Buehrer et. al. [BWS 05], SEM;

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



Attack Query

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

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

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

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

```
$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

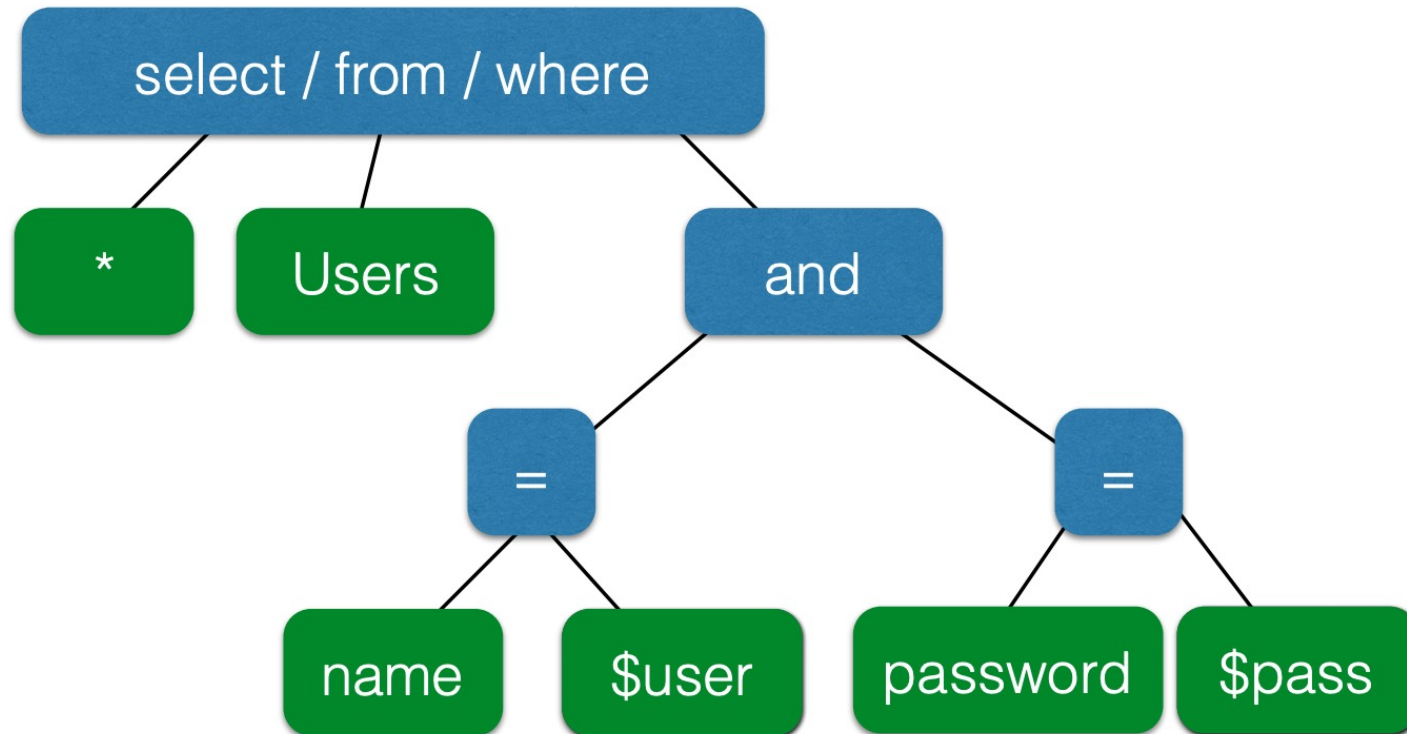
Decoupling let's us compile now, before binding the data

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

Bind Variables are typed

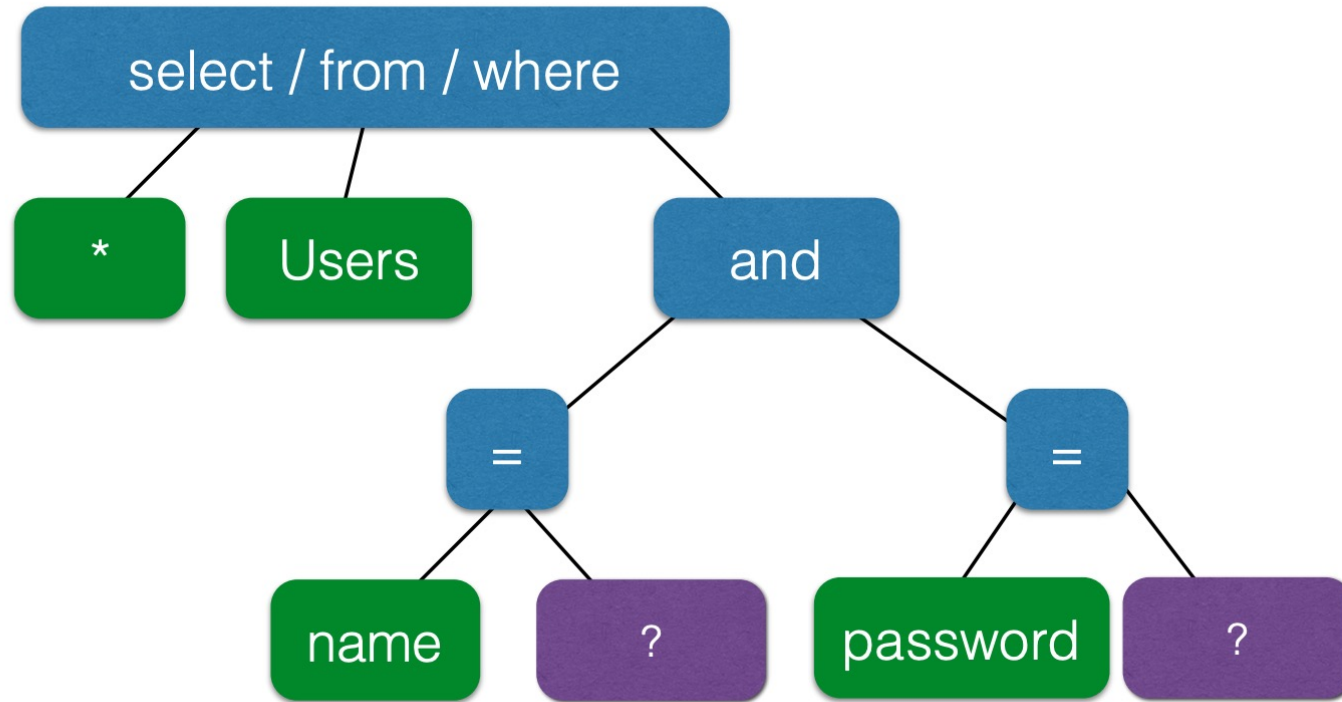
The underlying issue

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



The underlying issue

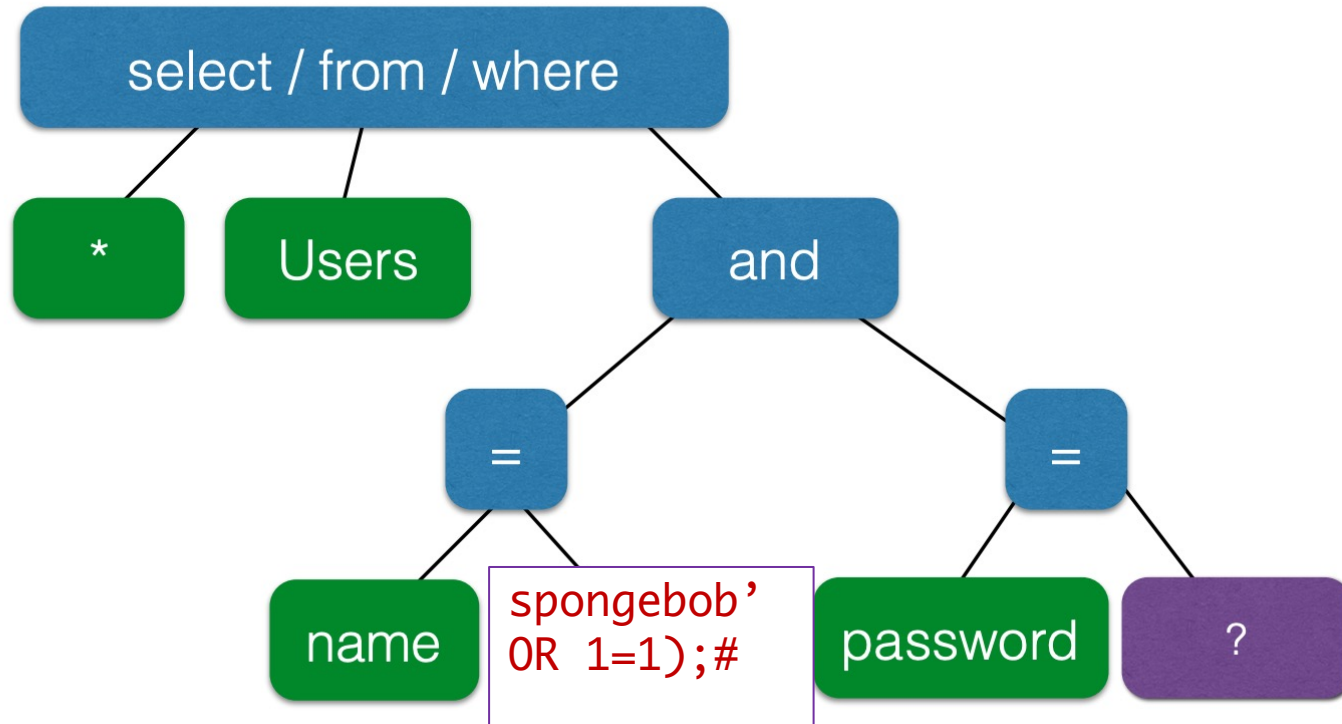
```
$statement = $db->prepare("select * from Users  
where(name=? and password=?);");
```



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

The underlying issue

```
$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