

Web Security

Anys Bacha

Slides from U. Shankar, M. Hicks, K. Du, D. Boneh, N. Zeldovich, A. Rahmati

SQL databases

- There are different kinds of DBMSes, differing in
 - organization of data
 - structure of transactions
 - etc.
- SQL DBMS are the most common
 - **SQL: Structured Query Language**
 - data is organized in **tables** (aka **relations**)
 - transactions work with the rows and columns of tables
 - Newer types of DBMSes
 - data remains unstructured
 - We're **not** looking at these

SQL Injection

SQL (Structured Query Language)

Users

Name	Gender	Age	Email	Password
Connie	F	12	connie@bc.com	j3i8g8ha
Steven	M	14	steven@bc.com	a0u23bt
Greg	M	34	greg@bc.com	0aergja
Cindy	F	35	cindy@bc.com	1bjb9a93

Get Records

```
SELECT *  
FROM Users  
WHERE name = 'Steven'
```

Update Records

```
Update Users  
SET email = 's@bc.com'  
WHERE name = 'Steven'
```

Insert Records

```
INSERT INTO Users  
Values('Ed', 'M',  
'15','ed@bc.com','pass123')
```

SQL (Structured Query Language)

Table

Users

Table name

Name	Gender	Age	Email	Password
Connie	F	12	connie@bc.com	j3i8g8ha
Steven	M	14	steven@bc.com	a0u23bt
Greg	M	34	greg@bc.com	0aergja
Cindy	F	35	cindy@bc.com	1bjb9a93

Row

(Record)

Column

SQL (Structured Query Language)

Users

Name	Gender	Age	Email	Password
Connie	F	12	connie@bc.com	j3i8g8ha
Steven	M	14	steven@bc.com	a0u23bt
Greg	M	34	greg@bc.com	0aergja
Cindy	F	35	cindy@bc.com	1bjb9a93

SELECT Age FROM Users WHERE Name='Greg';

SQL (Structured Query Language)

Users

Name	Gender	Age	Email	Password
Connie	F	12	connie@bc.com	j3i8g8ha
Steven	M	14	steven@bc.com	a0u23bt
Greg	M	34	greg@bc.com	0aergja
Cindy	F	35	cindy@bc.com	1bjb9a93

SELECT Age FROM Users WHERE Name='Greg'; **34**

SQL (Structured Query Language)

Users

Name	Gender	Age	Email	Password
Connie	F	12	connie@bc.com	j3i8g8ha
Steven	M	14	steven@bc.com	a0u23bt
Greg	M	34	mr.uni@bc.com	0aergja
Cindy	F	35	cindy@bc.com	1bjb9a93

SELECT Age FROM Users WHERE Name='Greg'; 34

UPDATE Users SET Email='mr.uni@bc.com'
WHERE Age=34; -- this is a comment

SELECT * FROM Users WHERE Age > 25
/* this is also a comment */ AND Gender='M';

SQL (Structured Query Language)

Users

Name	Gender	Age	Email	Password
Connie	F	12	connie@bc.com	j3i8g8ha
Steven	M	14	steven@bc.com	a0u23bt
Greg	M	34	mr.uni@bc.com	0aergja
Cindy	F	35	cindy@bc.com	1bjb9a93
Pearl	F	10000	pearl@bc.com	ziog9gga

SELECT Age FROM Users WHERE Name='Greg'; 34

UPDATE Users SET Email='mr.uni@bc.com'
WHERE Age=34; -- this is a comment

INSERT INTO Users Values('Pearl', 'F',...);

SQL (Structured Query Language)

Users

Name	Gender	Age	Email	Password
Connie	F	12	connie@bc.com	j3i8g8ha
Steven	M	14	steven@bc.com	a0u23bt
Greg	M	34	mr.uni@bc.com	0aergja
Cindy	F	35	cindy@bc.com	1bjb9a93
Pearl	F	10000	pearl@bc.com	ziog9gga

SELECT Age FROM Users WHERE Name='Greg'; 34

UPDATE Users SET Email='mr.uni@bc.com'

WHERE Age=34; -- this is a comment

INSERT INTO Users Values('Pearl', 'F',...);

DROP TABLE Users;

SQL (Structured Query Language)

SELECT Age FROM Users WHERE Name='Greg'; 34

UPDATE Users SET Email='mr.uni@bc.com'

WHERE Age=34; -- this is a comment

INSERT INTO Users Values('Pearl', 'F',...);

DROP TABLE Users;

Example 1

```
<?php
    $sql = "SELECT id, name, salary
            FROM credential
            WHERE eid='$eid'";
    $result = $conn->query($sql);
?>
```

If you do not know any eid, can you get the database to return some records?

eid

What do you put here?

Example 1

```
<?php
    $sql = "SELECT id, name, salary
            FROM credential
            WHERE eid='$eid'";
    $result = $conn->query($sql);
?>
```

If you do not know any eid, can you get the database to return some records?

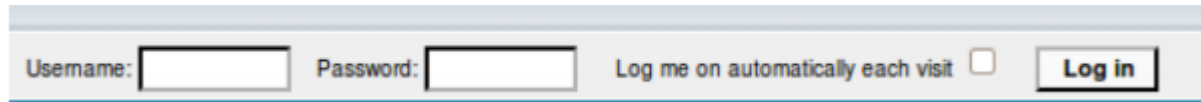
' XYZ' OR 1=1 -- '

eid

What do you put here?

Example 2

Website



Username: Password: Log me on automatically each visit ☐

“Login code” (PHP)

In class exercise

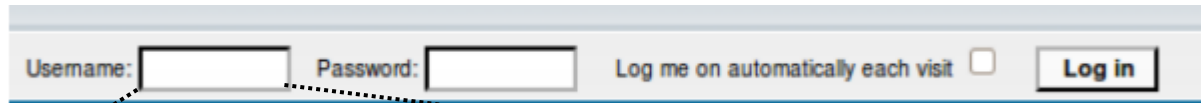
```
$sql = "select * from Users  
where name='$user' and password='$pass';  
$result = $conn->query($sql);
```

How do you log into this website?

How could you exploit this?

Example 2

Website



Username: Password: Log me on automatically each visit ☐

Frank' OR 1=1 --

```
$sql = "select * from Users  
where name='$user' and password='$pass';  
$result = $conn->query($sql);
```

```
$sql = "select * from Users  
where name='frank' OR 1=1 -- and  
password='whocares';
```

Login successful!

Problem: Data and code mixed up together

SQL injection: Worse

Website



Frank' OR 1=1; DROP TABLE Users --

```
$sql = "select * from Users  
where name='$user' and password='$pass';"  
$result = $conn->query($sql);
```

```
$sql = "select * from Users  
where name='frank' OR 1=1;  
DROP TABLE Users -- and password='whocares';"
```

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

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.

SQL Injection Countermeasures

The underlying issue

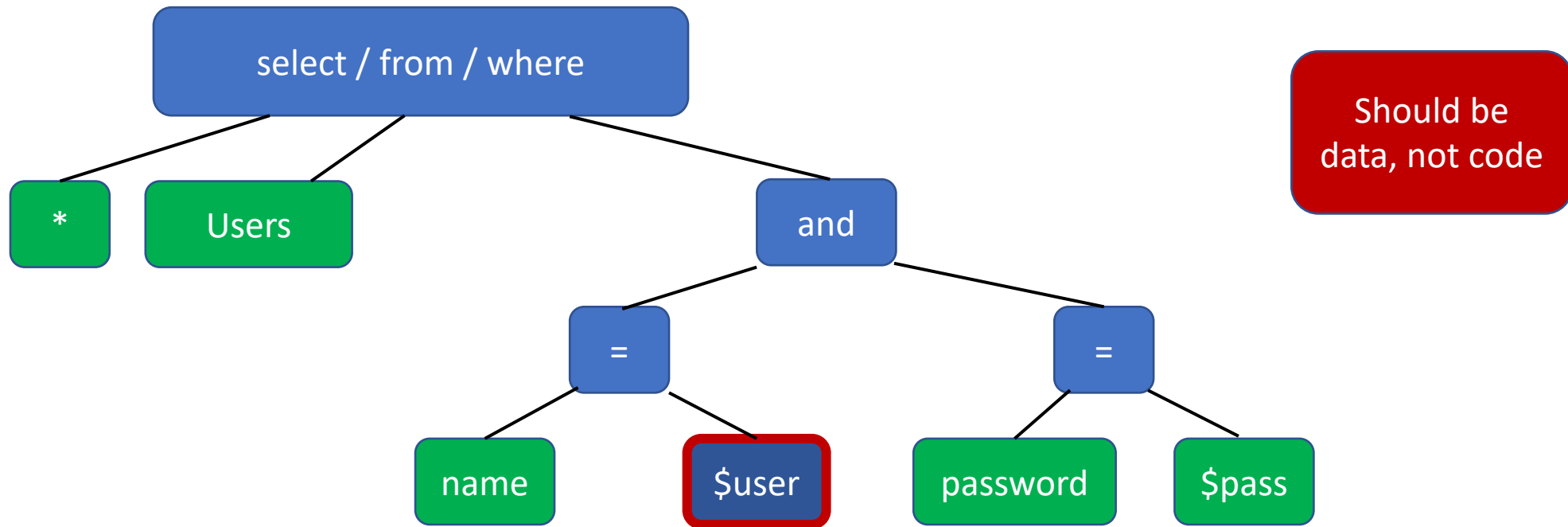
```
$sql = "select * from Users  
where name='$user' and password='$pass';  
$result = $conn->query($sql);
```

- 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

The underlying issue

```
$sql = "select * from Users  
where name='$user' and password='$pass';  
$result = $conn->query($sql);
```



Prevention: Input validation

- We require input of a certain form, but we cannot guarantee it has that form, so we must validate it
 - Just like we do to avoid buffer overflows
- Making input trustworthy
 - **Check** it has the expected form, reject it if not
 - **Sanitize** by modifying it or using it such that the result is correctly formed

Sanitization: Blacklisting

' ; --

- **Delete** the characters you don't want
- **Downside:** "Peter O'Connor"
 - You want these characters sometimes!
 - How do you know if/when the characters are bad?
- **Downside:** How to know you've ID'd all the bad characters

Sanitization: Escaping

- **Replace** problematic characters with safe ones
 - Change ' to \'
 - Change ; to \;
 - Change - to \-
 - Change \ to \\
- Hard by hand, there are many libs & methods
 - magic_quotes_gpc = On
 - mysql_real_escape_string()
- **Downside:** Sometimes you want these in your SQL

Checking: Whitelisting

- Check that the user input is **known to be safe**
 - E.g., integer within the right range
- Rationale: Given invalid input, **safer to reject than fix**
 - “Fixes” may result in wrong output, or vulnerabilities
 - Principle of fail-safe defaults
- **Downside:** Hard for rich input!
 - How to whitelist usernames? First names?

Sanitization via escaping, whitelisting,
blacklisting is HARD

Can we do better?

Sanitization: Prepared statements

- Treat user data according to its *type*
- Decouple the code and the data

```
$sql = "select * from Users  
where name='$user' and password='$pass'";  
$result = $conn->query($sql);
```

connect to DB

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

Prepare
statement

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

bind variables
to typed data

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

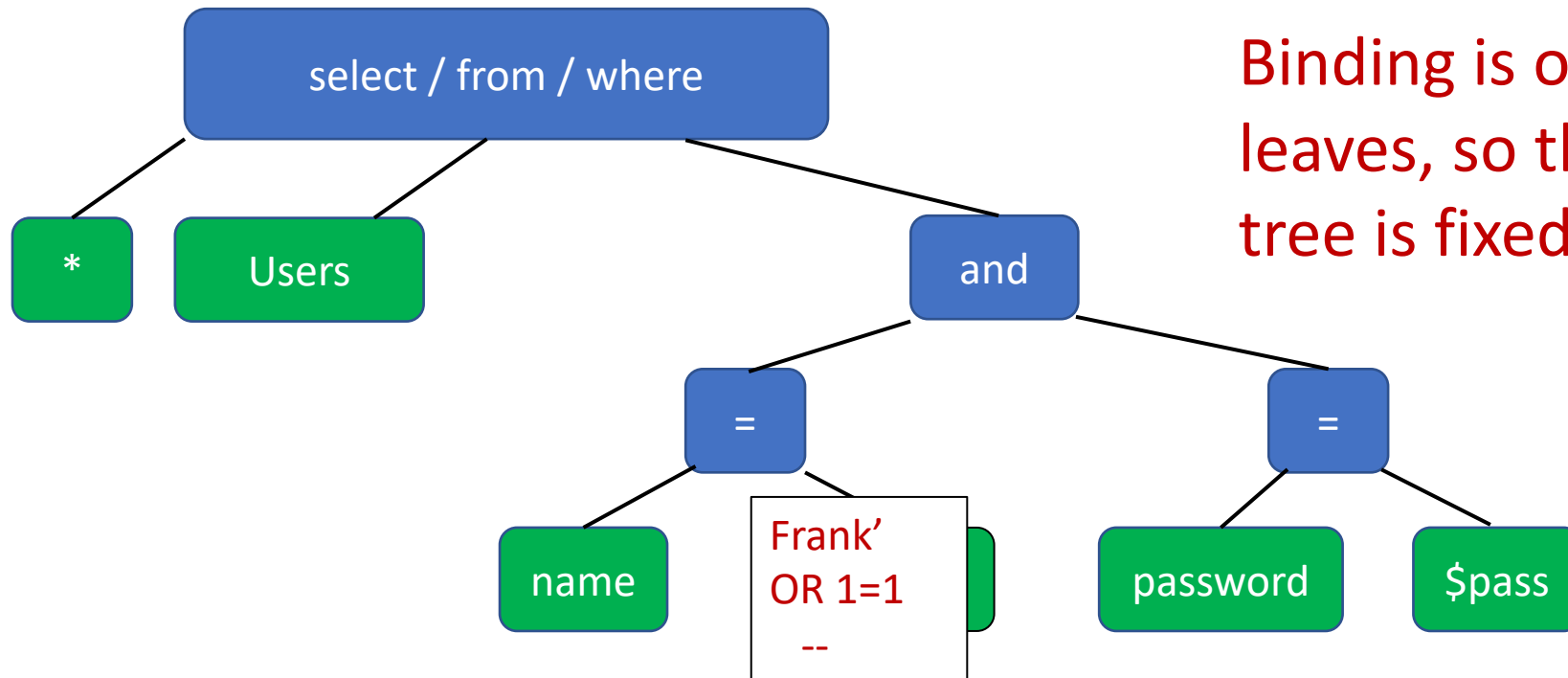
s for string
i for integer

```
$statement->execute();
```

**Decoupling lets us compile now,
before binding the data**

The underlying issue

```
$statement = $conn->prepare("select * from Users  
    where name=? and password=?");  
$statement->bind_param("ss", $user, $pass);
```



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

Additional mitigation

- For defense in depth, also try to mitigate any attack
 - Should always do input validation in any case!
- **Limit privileges**: reduces power of exploitation
 - Limit commands and/or tables a user can access
 - e.g., allow SELECT on Orders but not Creditcards
- **Encrypt sensitive data**: less useful if stolen
 - May not need to encrypt Orders table
 - But certainly encrypt **creditcards.cc_numbers**

SQL Injection Examples

- Check the following for SQLi examples:
 - <http://www.unixwiz.net/techtips/sql-injection.html>

END