Vulnerable SW - SQL Injection Attack

Information Security – Lecture 23

Aadil Zia Khan

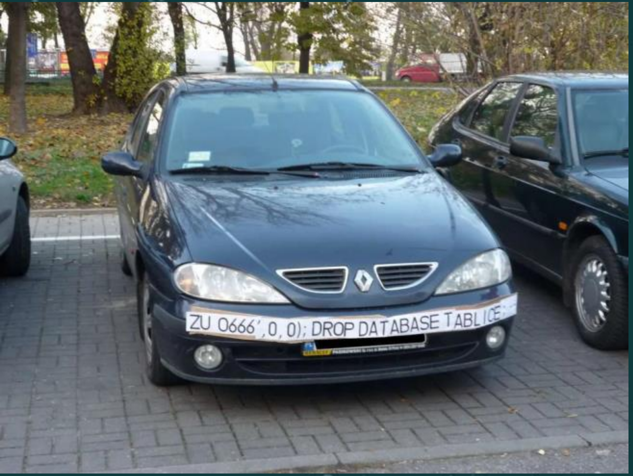
Can SQL Be Bad?

- Users upload information through forms on dynamic webpages
- Servers receive the information and place them inside an SQL query – which is then run
- Can the user upload a partial query instead of his own information???

SQL Injection

- SQL Injection attack is designed to send malicious SQL commands to the database server by placing malicious SQL statements in the user input
    - Attackers can dump database tables with hundreds of thousands of customer records
    - Attackers can also modify or delete data
    - Attackers can launch denial-of-service (DoS) attacks

SQL Injection



- Around 35% of applications affected by SQL Injection attacks - Veracode 2016 State of Software Security Report
- A single website received 94,057 SQL injection attack requests in one day - Imperva 2013
- A hacker replaced his license number with a partial SQL query and when the speed camera inserted this information into the database, the table was dropped

SQL Injection Attack

1. Application server sends form to user as an HTML page
2. Attacker inserts SQL exploit in some text field and submits the form
3. Application builds a query string with the exploit data
4. Application sends SQL query to the database
5. Database executes the query, including the exploit, and modifies the database (if a modifying query) and sends the resulting data (if any) back to the application
6. Application returns the data to the user

No need to worry about the firewalls protecting the server, because HTTP access to the server is already allowed

SQL Injection Attack Example - Numeric Field

Taken from w3schools

Assume backend code

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

Suppose, in the UserID field, the user enters

```
105 OR 1=1
```

Then, the SQL statement will look like this:

```
SELECT * FROM Users WHERE UserId = 105 OR 1=1;
```

The SQL above is valid and will return ALL rows from the "Users" table, since OR 1=1 is always TRUE.

Dangerous if the "Users" table contains private information

SQL Injection Attack Example - Text Field

Taken from w3schools

Assume backend code

```
uName = getRequestString("username");
uPass = getRequestString("userpassword");
sql = 'SELECT * FROM Users WHERE Name ="' + uName + '" AND Pass ="' + uPass + '"‘
```

Suppose, in the UserName and Password fields, the user enters

```
" OR ""="
```

Then, the SQL statement will look like this:

```
SELECT * FROM Users WHERE Name ="" or ""="" AND Pass ="" or ""=""
```

The SQL above is valid and will return all rows from the "Users" table, since OR ""="" is always TRUE

SQL Injection Attack Example - Batched SQL

Taken from w3schools

Assume backend code

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

Suppose, in the UserID field, the user enters

```
105; DROP TABLE Suppliers;
```

Then, the SQL statement will look like this:

```
SELECT * FROM Users WHERE UserId = 105; DROP TABLE Suppliers;
```

The SQL above is valid and will return row where UserID is 105, and then delete Suppliers table

SQL Injection Attack Example - Batched SQL

Taken from w3schools

For a string input, suppose the attacker enters

```
blah‘; DROP TABLE Suppliers; --
```

Then, the SQL statement will look like this:

```
SELECT * FROM Users WHERE Name = ‘blah’; DROP TABLE Suppliers; --’
```

comment (--) will cause the final quote to be ignored

Whatever the outcome of the first query, second would drop the table

Defense Techniques

- Object Relational Mappers
    - Developers can use ORM frameworks to create safe database queries
    - These queries are not strings, so there is no injection vulnerability

- Parameterized statements
    - E.g., => "SELECT * FROM Users WHERE UserId = @0";
    - Placeholder can only store a value of the given type which the engine checks
    - Hence the SQL injection would simply be treated as a strange (and probably invalid) parameter value

Defense Techniques

- Validate input
  - Email addresses, dates, part numbers, etc.
  - Verify that the input is a valid string in the language
  - Problematic characters (e.g., ‘*’) may be prohibited
  - Exclude quotes and semicolons

- Have length limits on input

Defense Techniques

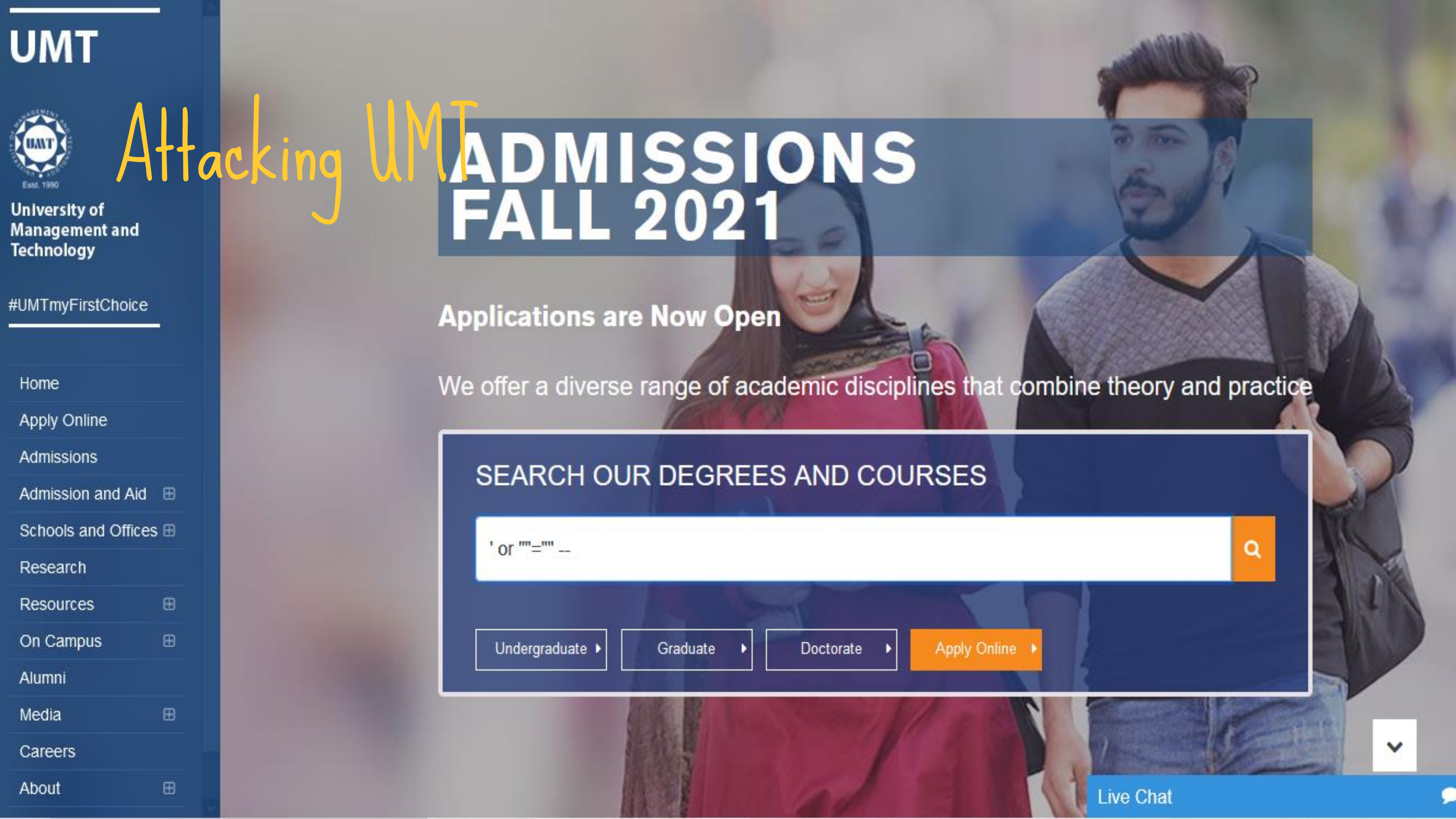
- Remove SQL keywords from query string
  - INSERT, DROP, etc.
  - Can further check to see if they represent a statement or valid user input

- Limit database permissions and segregate users
  - If user only needs read permission, don't grant write permissions

Defense Techniques

- Hide error reporting
  - Default error reporting often gives away information that is valuable for attackers (table name, field name, etc.)
  - Configure so that this information is never exposed to a user

Attacking UMT



UMT

Estd. 1990

University of Management and Technology

#UMTmyFirstChoice

- Home
- Apply Online
- Admissions
- Admission and Aid ⊞
- Schools and Offices ⊞
- Research
- Resources ⊞
- On Campus ⊞
- Alumni
- Media ⊞
- Careers
- About ⊞

ADMISSIONS FALL 2021

Applications are Now Open

We offer a diverse range of academic disciplines that combine theory and practice

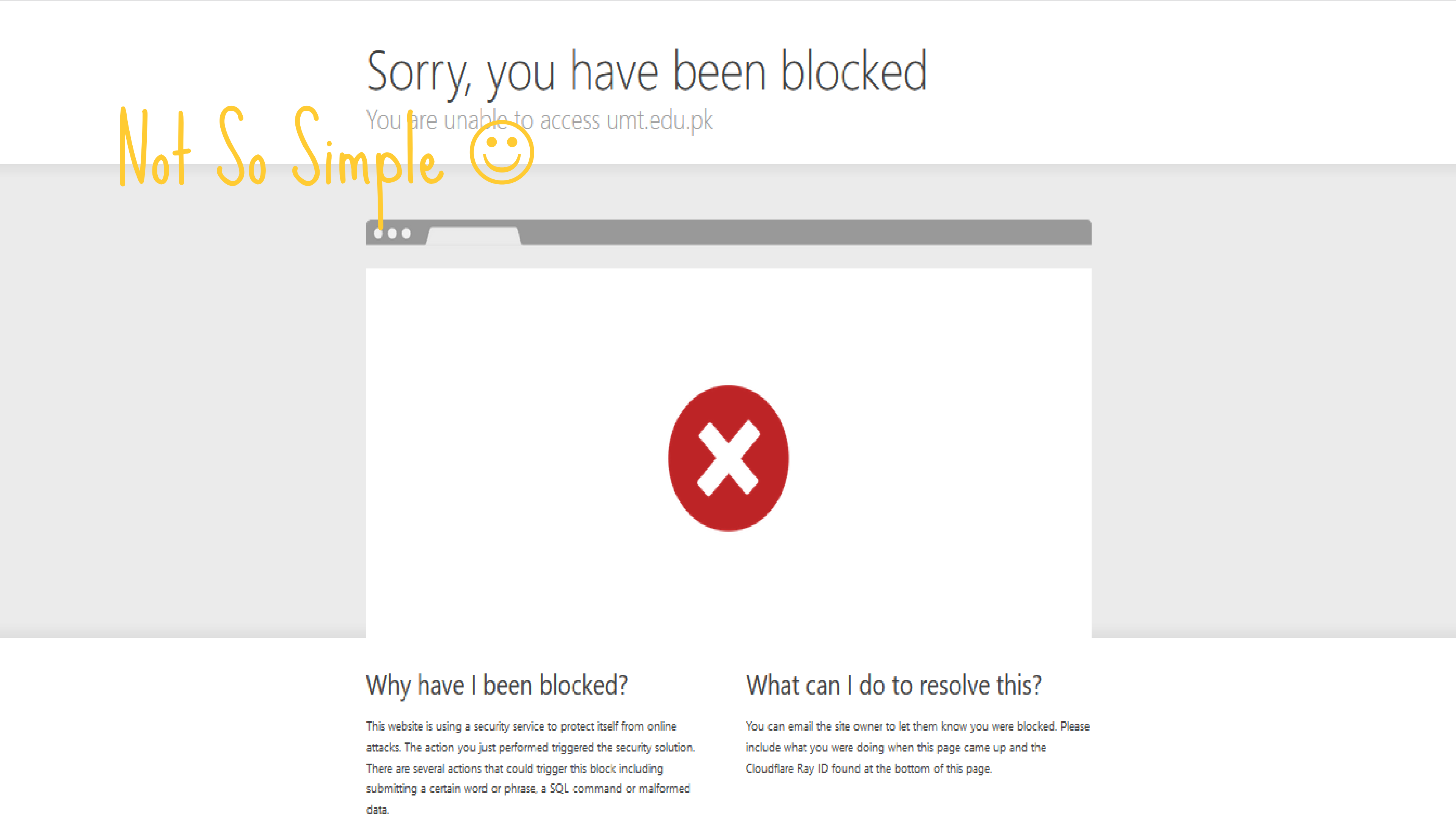
SEARCH OUR DEGREES AND COURSES

```
' or ""="" --
```

Undergraduate ▸ Graduate ▸ Doctorate ▸ Apply Online ▸

Live Chat

Not So Simple ☺



Sorry, you have been blocked

You are unable to access umt.edu.pk

Why have I been blocked?

This website is using a security service to protect itself from online attacks. The action you just performed triggered the security solution. There are several actions that could trigger this block including submitting a certain word or phrase, a SQL command or malformed data.

What can I do to resolve this?

You can email the site owner to let them know you were blocked. Please include what you were doing when this page came up and the Cloudflare Ray ID found at the bottom of this page.