Scaling

Replication and Redundancy

- Redundancy:
  - Multiple copies of same data
  - Often used in connection with backups - even if one fails, others survive
  - One copy is still the master
- Replication:
  - Usually in context of performance
  - May not be for purpose of backup
  - Multiple sources of same data - less chance of server overload
  - Live replication requires careful design

BASE vs ACID

- “Basically Available”, “Soft state”, “Eventually consistent”
  - Winner of worst acronym award
- Eventual consistency instead of Consistency
  - Replicas can take time to reach consistent state
- Stress on high availability of data

Replication in traditional DBs

- RDBMS replication possible
- Usually server cluster in same data center
  - Load balancing
- Geographically distributed replication harder
  - Latency constraints for consistency

Scale-up vs Scale-out

- Scale-up: traditional approach
  - Larger machine
  - More RAM
  - Faster network, processor
  - requires machine restart with each scale change
- Scale-out:
  - Multiple servers
  - Harder to enforce consistency etc. - better suited to NoSQL / non-ACID
  - Better suited to cloud model: Google, AWS etc provide automatic scale-out, cannot do auto-scale-up

Application Specific

- Financial transactions:
  - cannot afford even slightest inconsistency
  - Only scale-up possible
- Typical web-application
  - Social networks, media: eventual consistency OK
  - e-commerce: only the financial part needs to go to ACID DB

Security

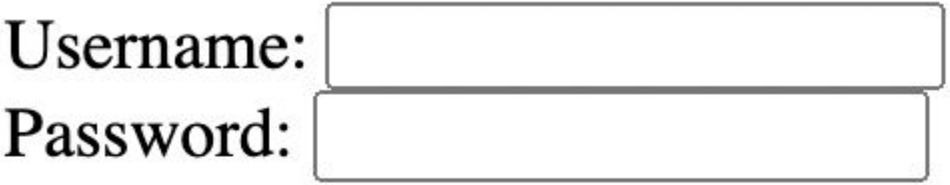
SQL in context of an application

- Non-MVC app: can have direct SQL queries anywhere
- MVC: only in controller, but any controller can trigger a DB query

So what’s dangerous about queries?

Typical HTML form

```
<form>
   Username:
   <input type="text" name="name">
   <br />
   Password:
   <input type="password" name="password">
   <br />
</form>
```



Username:

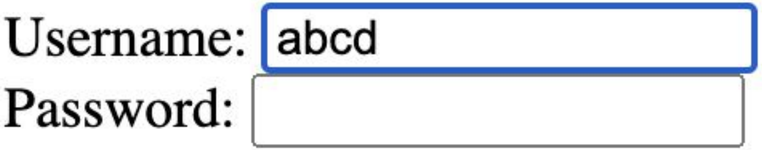
Password:

Code

```
name = form.request["name"]
pswd = form.request["name"]

sql = 'SELECT * FROM Users WHERE Name ="' +
       name + '" AND Pass ="' + pswd + '"'
```

Example input vs SQL

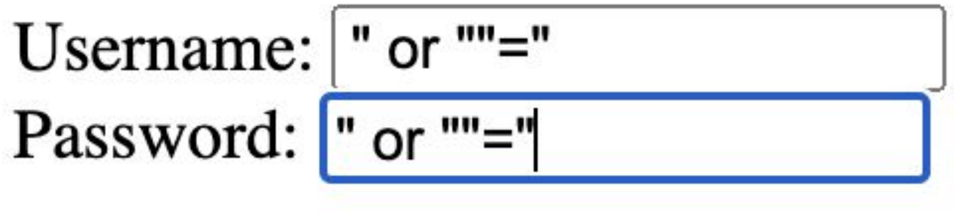


Username: abcd

Password:

```
SELECT * FROM Users WHERE Name ="abcd" AND Pass = "pass"
```

Example input vs SQL



Username: " or ""="

Password: " or ""="

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

Result???

Example input vs SQL

```
sql = "SELECT * FROM Users WHERE Name = " + name
```

```
Input:

a; DROP TABLE Users;

Query:

SELECT * FROM Users WHERE Name = a; DROP TABLE Users;
```

Problem

- Parameters from HTML taken without validation
- Validation:
  - Are they valid text data (no special characters, other symbols)
  - No punctuation or other invalid input
  - Are they the right kind of input (text, numbers, email, date)?
- Validation **MUST** be done just before the database query - even if you have validation in the HTML or Javascript - not good enough
  - Direct HTTP requests can be made with junk data

Web Application Security

- SQL injection
  - Use known frameworks, best practices, validation
- Buffer overflows, input overflows
  - Length of inputs, queries
- Server level issues - protocol implementation?
  - Use known servers with good track record of security
  - Update all patches
- Possible outcomes:
  - loss of data - deletion
  - exposure of data - sensitive information leak
  - manipulation of data - change

HTTPS ?

- Secure sockets: secure communication between client and server
- Server certificate:
  - based on DNS: has been verified by some trusted third party
  - difficult to spoof
  - based on Mathematical properties - ensure very low probability of mistakes match
- However:
  - Only secures link for data transfer - does not perform validation or safety checks
  - Negative impact on “caching” of resources like static files
  - Some overhead on performance

Summary

- Internet and Web security are complex: enough for a course in themselves
- Generally recommended to use known frameworks with trusted track records
- Code audits
- Patch updates on OS, server, network stack etc. essential

App developers should be very careful of their code, but also aware of problems at other levels of the stack