# Administrivia

HW3 due 10/29

HW4, Project 2 released next week

COLUMBIA UNIVERSITY
DATA SCIENCE INSTITUTE

**Special Seminar: A Survey of Cloud Database Systems with C. Mohan**

**Monday, October 28, 2024**
**12:00 pm - 1:00 pm**

# Lecture 14
# Security and Privacy

# Security

SQL Injection

Privacy vs Security

Access Controls and GRANT

Encryption


FEATURES | By Joseph Cox | Nov 20 2015, 8:00am

# The History of SQL Injection, the Hack That Will Never Go Away

**Over 15 years after it was first publicly disclosed, SQL injection is still the number one threat to websites.**

https://motherboard.vice.com/en_us/article/aekzez/the-history-of-sql-injection-the-hack-that-will-never-go-away

sql injection

FILTER


## Running an SQL Injection Attack - Computerphile

Computerphile 1.6M views • 2 years ago

Just how bad is it if your site is vulnerable to an **SQL Injection**? Dr Mike Pound shows us how they work. Cookie Stealing: ...


## SQL Injection Attack Tutorial (2019)

HackHappy • 76K views • 8 months ago

**SQL Injection** attacks are still as common today as they were ten years ago. Today I'll discuss what are SQLi and how you can ...

CC


## SQL Injection Basics Demonstration

Imperva • 360K views • 9 years ago

Imperva presents an educational video series on Application and Database Attacks in High Definition (HD)


## Hacking Websites with SQL Injection - Computerphile

Computerphile 1.4M views • 5 years ago

Websites can still be hacked using **SQL injection** - Tom explains how sites written in PHP (and other languages too) can be ...

CC


## DEFCON 17: Advanced SQL Injection

Christiaan008 • 220K views • 8 years ago

Speaker: Joseph McCray Founder of Learn Security Online **SQL Injection** is a vulnerability that is often missed by web application ...

# SQL Injection

Pass *sanitized* values to the database

```
args = ('Dr Seuss', '40')
conn1.execute(
    "INSERT INTO users(name, age) VALUES(%s, %s)",
    args)
```

Pass in a tuple of query arguments

DBAPI library will *properly escape* input values

Most libraries support this

*Never construct raw SQL strings*

# SQL Injection

## Why pass values using query parameters?

```
name = “eugene”

conn1.execute(
  “SELECT * FROM users WHERE name=%s”, name)
```

```
conn1.execute(
  “SELECT * FROM users WHERE name=‘{name}’”.format(name=name))

   SELECT * FROM users WHERE name=‘eugene’
```

# SQL Injection

## Why pass values using query parameters?

```
name = "eugene';\nDELETE * FROM users;--"

conn1.execute(
  "SELECT * FROM users WHERE name=%s", name)
```

```
conn1.execute(
  "SELECT * FROM users WHERE name='{name}'".format(name=name))
```

```
SELECT * FROM users WHERE name='eugene';
DELETE * FROM users;
--'
```

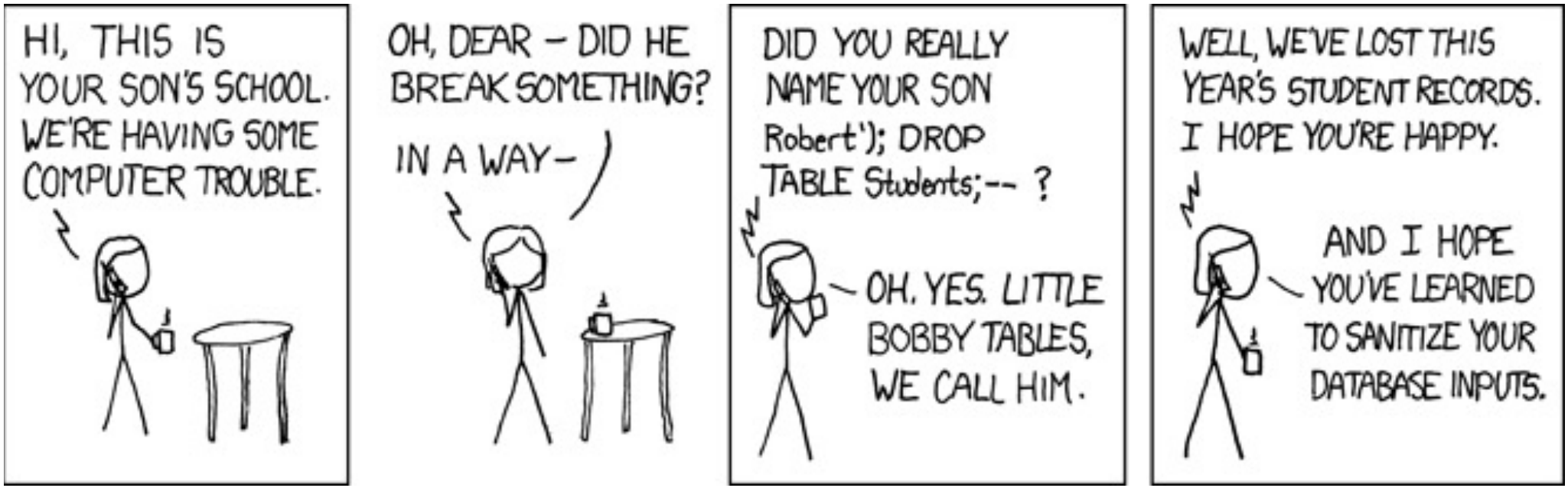
# SQL Injection

http://w4111.github.io/inject

code on github:
`w4111/w4111.github.io/src/injection/`

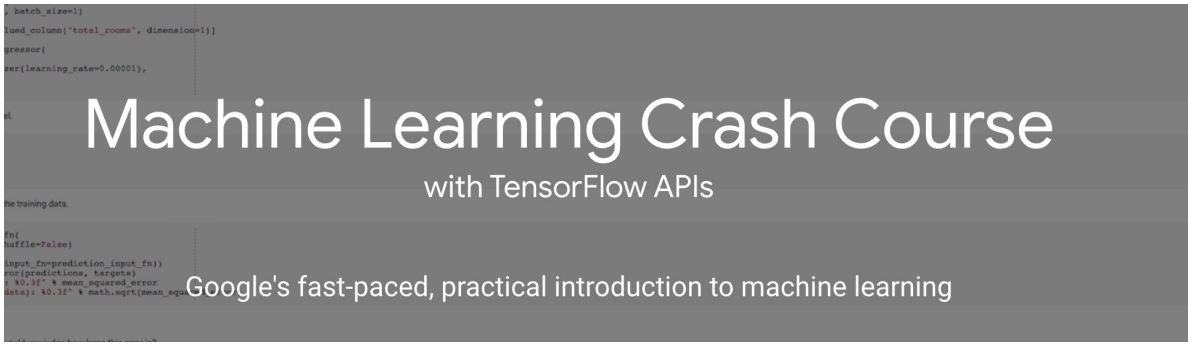
Use query parameters by passing form values as arguments to `execute()`
Library sanitizes inputs automatically (and correctly!)

# SQL Injection




Project: You'll need to protect against simple SQL injections

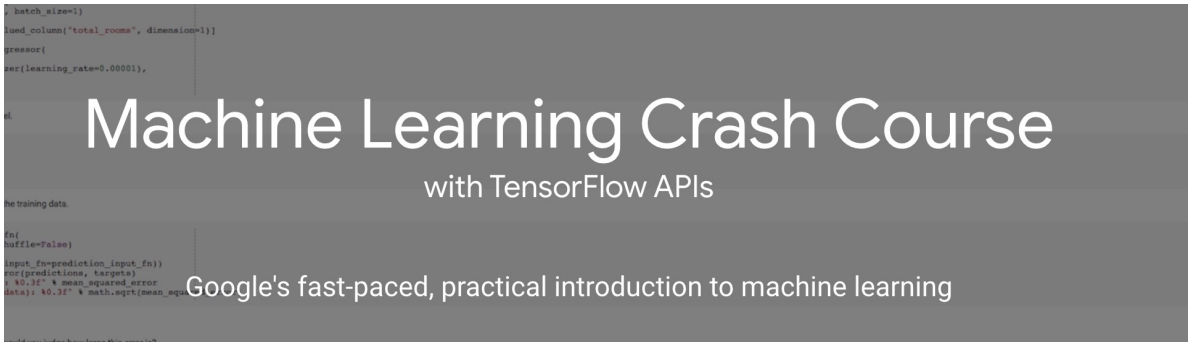
More examples: https://corenumb.wordpress.com/2016/05/14/mr-robot-blind-sql-injection-vulnerability/




**If you had to prioritize improving one of the areas below in your machine learning project, which would have the most impact?**

- A more clever loss function
- A deeper network
- The quality and size of your data
- Using the latest optimization algorithm

https://developers.google.com/machine-learning/data-prep




**If you had to prioritize improving one of the areas below in your machine learning project, which would have the most impact?**

A more clever loss function

A deeper network

The quality and size of your data ✓

Data trumps all. It's true that updating your learning algorithm or model architecture will let you learn different types of patterns, but if your data is bad, you will end up building functions that fit the wrong thing. The quality and size of the data set matters much more than which shiny algorithm you use.

**Correct answer.**

Using the latest optimization algorithm

# Privacy vs Security

Privacy:

- person's right to control their personal information and how it's used
- Should not release data that can be used to *infer* user's personal information
- hard to define

Security:

- prevent unauthorized access to data
- access controls
- encryption prevents reading data even w/ access

# Access Control and GRANT

Different user accounts in w4111 DB

- staff, student, ew2493

Each user only has access privileges to read/modify subset of DB:

- Privileges: read, insert, update, delete
- Objects: Databases, Schemas, Tables, Views, Attributes
- Who? users, roles

https://www.postgresql.org/docs/current/sql-grant.html

# Access Control and GRANT

```
GRANT <privileges>
   ON <objects>
   TO <users/roles>
```

```
CREATE USER ew2493;
CREATE ROLE admin;

GRANT SELECT, INSERT ON users TO admin;     // users relation
GRANT CREATE, CONNECT ON DATABASE test TO admin;

GRANT admin to ew2493;
```

# Access Control and GRANT

How to restrict user's access to subsets of a relation or only aggregated statistics?

Combine GRANT and Views!

```
CREATE VIEW stats AS
    SELECT department, avg(salary)
      FROM costs
  GROUP BY department

GRANT SELECT ON stats TO appuser
```

# Hashing and Encryption

Prevent data from being read, even if database contents are accessed

**Hashing**: one way function, loses original data

- used to check equality

```
Data → hash() → hasheddata
hash(password) == hash(input)
```

**Encryption**: 2 way function, is reversible

- only users with key can read
- key needs to be kept safe!

```
Data → encrypt(key) → encdata → decrypt(key) → Data
```

https://www.postgresql.org/docs/current/encryption-options.html

# Hashing and Encryption

DBMS support varies

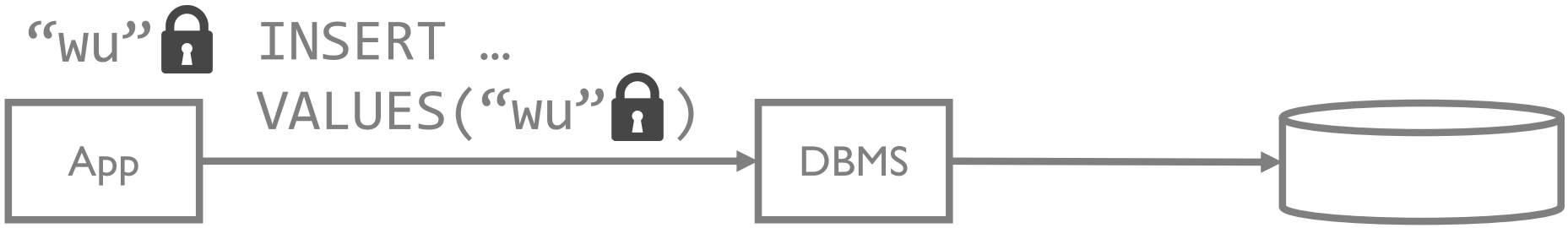
Hashing implemented as UDFs

- `INSERT INTO users VALUES(name, hash(password))`

Encryption

- encrypted hard drive, encrypted disk blocks, table, columns, …

Application encrypts `data` before issuing queries



# Hashing and Encryption

DBMS support varies

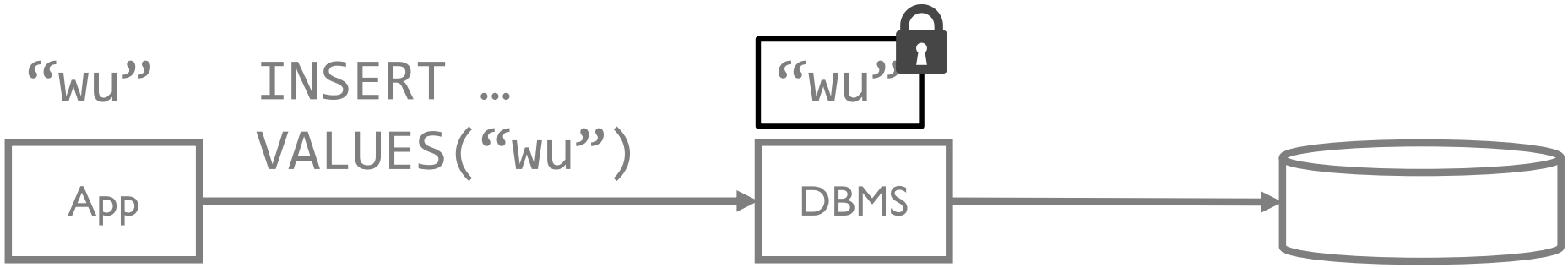
Hashing implemented as UDFs

- `INSERT INTO users VALUES(name, hash(password))`

Encryption

- encrypted hard drive, encrypted disk blocks, table, columns, …

DBMS encrypts received data (granularity of cells, rows, tables, or DB)



# Hashing and Encryption

DBMS support varies

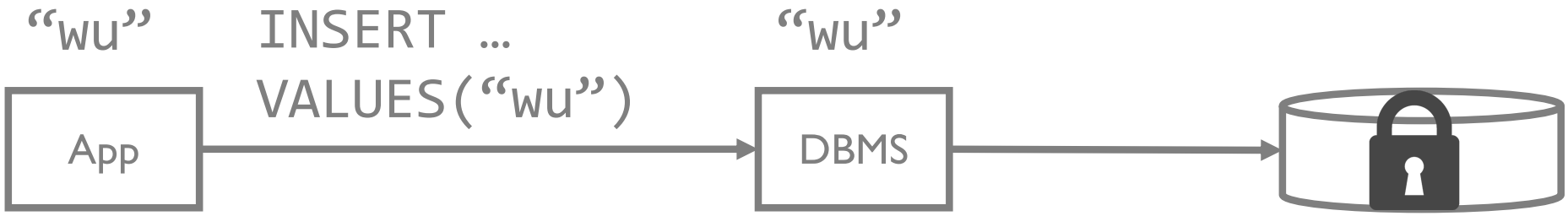
Hashing implemented as UDFs

- `INSERT INTO users VALUES(name, hash(password))`

Encryption

- encrypted hard drive, encrypted disk blocks, table, columns, …

Storage encrypts everything on e.g., hard drive



# Hashing and Encryption

DBMS support varies

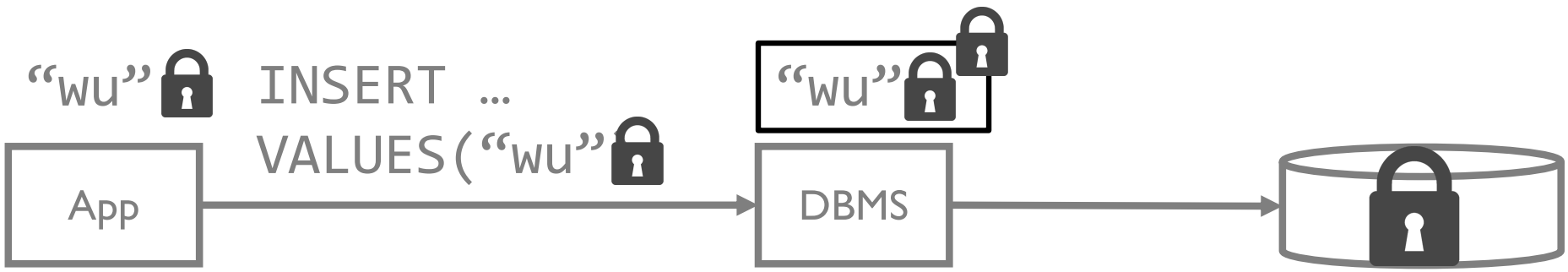
Hashing implemented as UDFs

- `INSERT INTO users VALUES(name, hash(password))`

Encryption

- encrypted hard drive, encrypted disk blocks, table, columns, …

Can mix and match



# What to Understand

Why Embedded SQL is no good

Client-server vs embedded DBMSes

DBAPI components, cursors

Impedance mismatch: examples and possible solutions

SQL injection and protections

Levels of access controls and Views

Conceptual understanding of privacy & encryption