INF1343, Winter 2011

Data Modeling and Database Design

Yuri Takhteyev
University of Toronto

Week 10

Database Security

Final Project

Implementation

- plan for unexpected problems
- the database is what matters most
- backup your stuff
- test!

Documentation

- update the ER in the end
- include all SQL (even if it doesn't work)
- explain what is not obvious design
- explain non-obvious testing steps

Questions?

Security

Confidentiality / secrecy:

Preventing unwanted disclosure

Integrity

Preventing unwanted modification

Availability

Preventing disruption or loss

(Consider the interrelation between the 3.)

Threats

Attacks

- Profit
- Revenge/terrorism
- Lulz

Mistakes / accidents / oversights

Outsiders vs insiders

- Insiders are armed with internal knowledge.
- Consider “social engineering”.

Access Control

Define who can do/see what

1. Identifying users
   - Passwords + other mechanisms
   - Users can be assigned to groups/roles
2. Determining rights
   - Grant/deny rights to specific data/operations
   - Could be done at the level of group/role

SQL Privileges

```
GRANT «privilege» ON «object»
TO «user»;
```

For instance:

```
GRANT SELECT ON secretdb.*
TO kenobio1;
```

Variations:

```
GRANT SELECT ON secretdb.*
TO kenobio1 WITH GRANT OPTION;
```

SQL Privileges

```
REVOKE «privilege» ON «object»
FROM «user»;
```

For instance:

```
REVOKE SELECT ON secretdb.*
FROM kenobio1;
```

Also (not in MySQL):

```
REVOKE SELECT ON secretdb.*
FROM kenobio1 CASCADE;
```

SQL Privileges

Variations:

```
GRANT SELECT ON secretdb.alpha
TO kenobio1;
```

The user can see table “alpha” but not the other tables.

Roles

More flexible:

```
CREATE ROLE student;
GRANT SELECT ON secretdb.*
TO student;
GRANT student TO kenobio1;
```

(Not available in MySQL.)

Using Views

```
CREATE TABLE true_omega
(boring TEXT, scary TEXT);

CREATE VIEW omega AS
SELECT boring from true_omega;

GRANT ALL ON secretdb.omega
TO kenobio1;
```

The user can now see and modify the values in “boring” but not in “scary”.

Using Views

```
CREATE VIEW omega AS
SELECT boring from true_omega
WHERE scary IS NULL;

GRANT ALL ON secretdb.omega
TO kenobio1;
```

The user can now see and modify the values in “boring” but only for those roles for where “scary” is not set.

Limitations

Somewhat inflexible

- 1000 employees = 1000 views?
- Solution: "fine-grained access control"

Discretionary

- Can be bypassed by users with access:
  - Obvious: "with grant option"
  - Less obvious: users with clearance can copy sensitive data into tables accessible to users without clearance
- Solution: "mandatory access control"
  - i.e., a centrally enforced row-level policy

Creating Users

```
CREATE USER alice;
SET PASSWORD FOR alice
= PASSWORD('bob');
```

You can now login as 'alice' with password 'bob':

```
mysql -p -u alice
```

BTW, to change your own password:

```
SET PASSWORD = PASSWORD('newpass');
```

Storing User Data

```
USE mysql;

select User, Password
from user where User="alice";
```

You don't have access to database “mysql,” but here is what I get to see:

```
+-------+-------------------------------------------+
| User  | Password                                  |
+-------+-------------------------------------------+
| alice | *61584B76F6ECE8FB9A328E7CF198094B2FAC55C7 |
+-------+-------------------------------------------+
1 row in set (0.00 sec)
```

Hashing Passwords

```
SET PASSWORD FOR alice
= PASSWORD('bob');
```

Compare with:

```
SELECT PASSWORD('bob');
SELECT SHA1('bob');
```

Hashing Passwords

A "one-way" function

bob → 61584B76F6ECE8FB9A328E7CF198094B2FAC55C7

Easily calculated.

61584B76F6ECE8FB9A328E7CF198094B2FAC55C7 → bob

Very hard to calculate (in theory).

How hard is "very hard"?

MD5: a few seconds
SHA1/2: 1 hour + $2
SHA3: hopefully centuries

Password Attacks

A brute-force attack

- Just try all possible combinations
  - 5 lower case letters: 12 million combinations
  - 5 letters / numbers: ~ 1 billion
  - 8 letters / numbers: > 200 trillion

A dictionary attack

- Check passwords based on words

Matching passwords to accounts

- Who uses “123456” as their password?

Password Trilemma

Pick a simple one

- Might be guessed or brute-forced

Write it down

- Might be found

Forget it later

- Need a procedure for recovery

Password Theft

Stored passwords

Paper, email accounts, files

Insecure transmission

“Sniffing”, “key-loggers”

Get a user to reveal the password

Phishing, social engineering

Password Alternatives

Public key cryptography

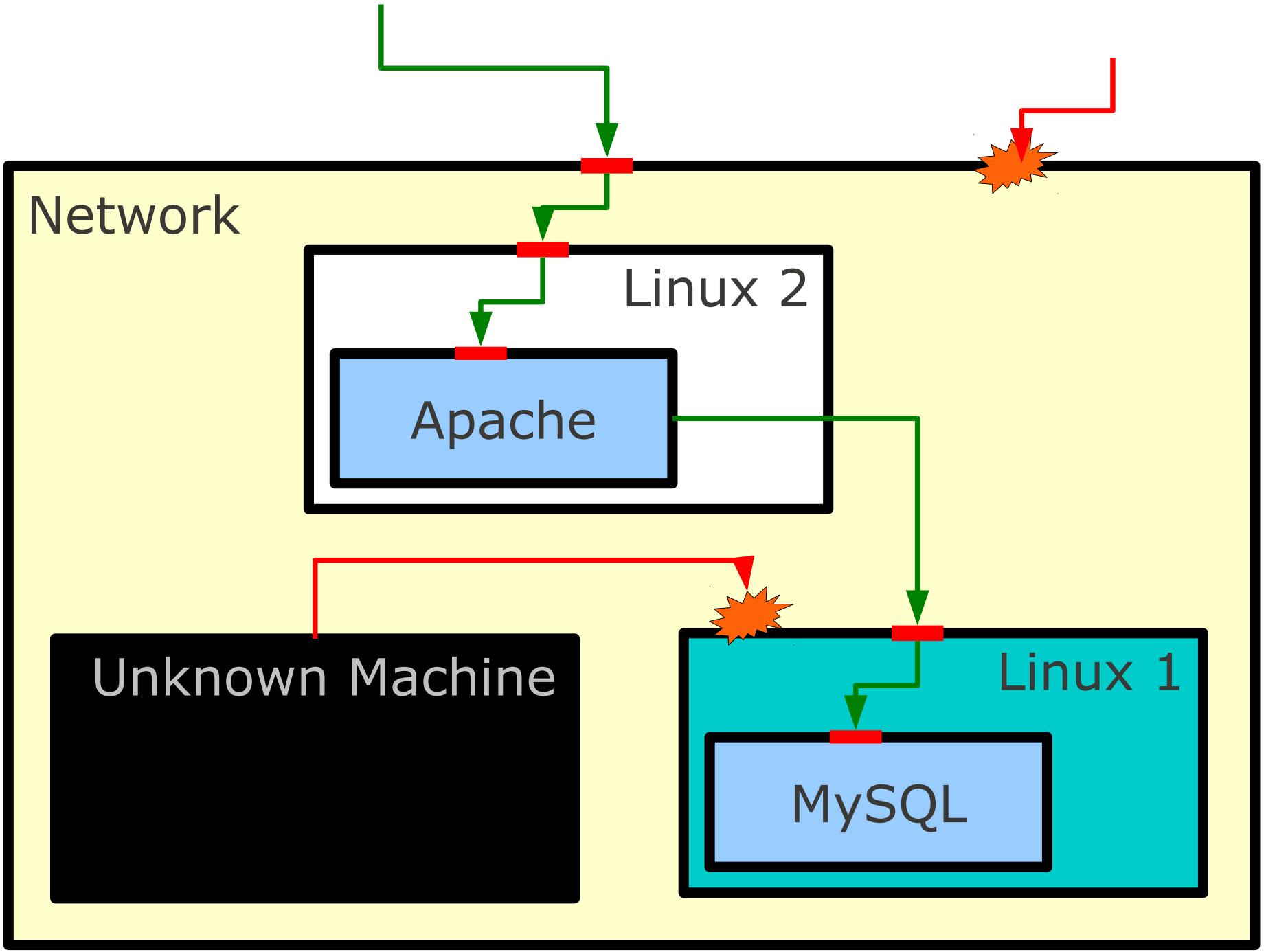
Authentication *without* revealing any secrets

A pair of keys (long numbers):

- A public key for encoding
- A private key for decoding

RSA SecurID:

image source: http://en.wikipedia.org/wiki/File:SecureID_token_new.JPG


Network
Linux 2
Apache
Unknown Machine
Linux 1
MySQL

Web App Accounts

Create a table for user accounts:

- usernames
- hashed passwords
  - never store actual passwords!

Authenticate the user with a query

```
select * from users where
username="%s" AND password="%s";
```

Proceed if successful

- Give the user a "cookie" for the future
  - cookies can be stolen (unless encrypted w/ SSL)

The Accounts

MySQL accounts

to access the database
(from an authorized machine)

Machine accounts

to access a machine
(from within the network)

Application accounts

your users, e.g. bank clients

Gawker Media

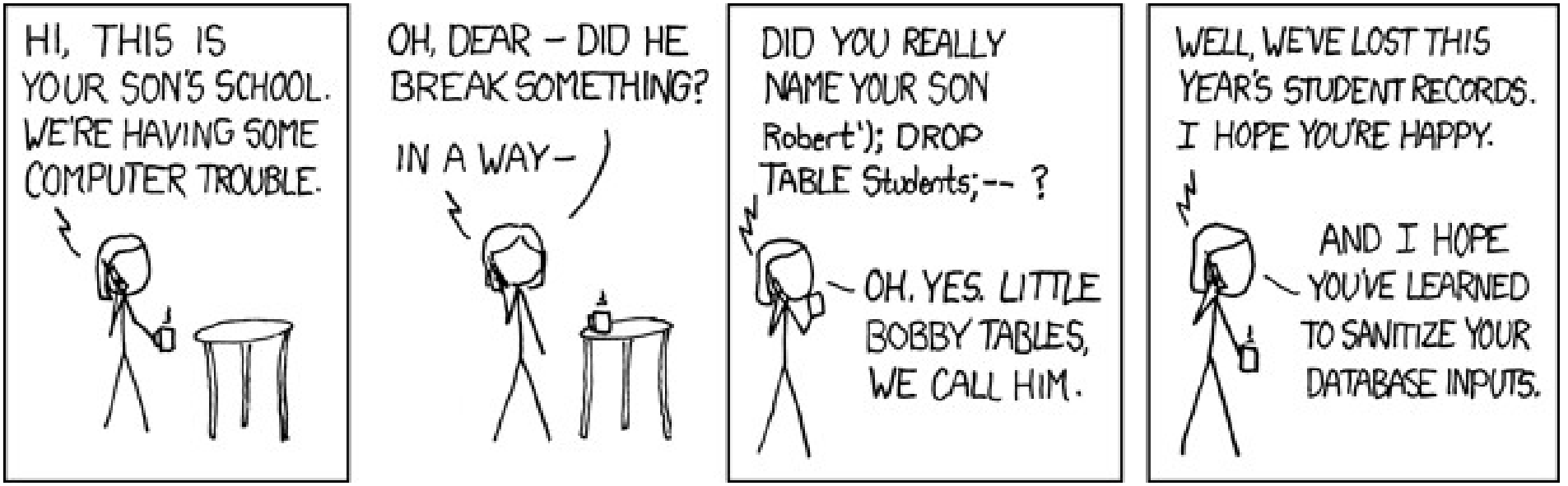
Hacked by “Gnosis” in 2010

- 1.25 million accounts stolen.
- 300,000 weakly hashed passwords released.
  - (Most common ones: “12345”, “password”)

The scenario:

hacked email → access to chat logs chat →
→ sysadmin passwords → access to servers →
→ root access (unpatched OS) →
→ MySQL access →
→ MySQL data, including passwords →
→ Twitter logins, etc. → Twitter spam

SQL Injection



Did you really name your son
Robert'); DROP TABLE Students;--?

Image source: XKCD, http://xkcd.com/327/

Dynamic SQL (Wrong)

```
TEMPLATE = """select * from persona
where species='%s';"""
import cgi
form = cgi.FieldStorage()
if form.has_key("species") :
  cursor=db.cursor()
  species = form["species"].value
  query =  TEMPLATE % species
  print query
  cursor.execute(query);
  rows = cursor.fetchall()
  print "<table>"
  for row in rows:
    print "<tr>"
    for x in [0,1,2]:
      print "<td>", row[x], "</td>"
    print "</tr>"
  print "</table>"
```

Don’t do this!

Here Is Why

http://yoda.ischool.utoronto.ca/~kenobio1/cgi-bin/personas2.cgi?species=Human'%3Binsert+into+persona+(name,species)+values+('X','Human')%3Bselect+*+from+persona+where+'

Dynamic SQL - Right

```
TEMPLATE = """select * from persona
where species='%s';"""
import cgi
form = cgi.FieldStorage()
if form.has_key("species") :
  cursor=db.cursor()
  species = form["species"].value
  query = TEMPLATE % db.escape_string(species)
  print query
  cursor.execute(query);
  rows = cursor.fetchall()
  print "<table>"
  for row in rows:
    print "<tr>"
    for x in [0,1,2]:
      print "<td>", row[x], "</td>"
    print "</tr>"
  print "</table>"
```

Anonymous vs HBGary

This domain has been seized by Anonymous under section #14 of the rules of the Internet.

Greetings HBGary (a computer "security" company),

Your recent claims of "infiltrating" Anonymous amuse us, and so do your attempts at using Anonymous as a means to garner press attention for yourself. How's this for attention?

You brought this upon yourself. You've tried to bite at the Anonymous hand, and now the Anonymous hand is bitch-slapping you in the face. You expected a counter-attack in the form of a verbal braul (as you so eloquently put it in one of your private emails), but now you've received the full fury of Anonymous. We award you no points.

(An SQL injection.)

Cablegate

260,000 “cables”

> “I would come in with music on a CD-RW labelled with something like ‘Lady Gaga’… erase the music … then write a compressed split file. No one suspected a thing … [I] listened and lip-synched to Lady Gaga’s Telephone while exfiltrating possibly the largest data spillage in American history.”

Solutions?

Physical Attack

http://bit.ly/server_theft

(Also: accidents, natural disasters, hardware failure, file corruption.)

Backup

mysqldump

```
mysqldump –u kenobio1 –p starwars > starwars.sql
mysql < starwars.sql
```

(Add “create database starwars; use starwars;”)

hot backup

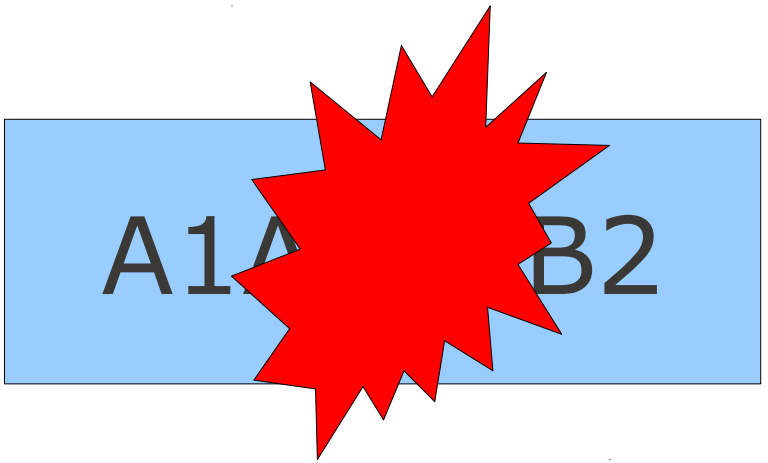
replication

off-site copies

RAID

combining disks for reliability and/or performance

No RAID



RAID-0

can be faster, no gain in reliability

RAID-1

faster reading, slower writing
can withstand losing one disk

RAID >2

RAID >2

reasonable reliability,
not too wasteful

Questions?