Web security

With material from Dave Levin, Mike Hicks, Lujo Bauer

Previously

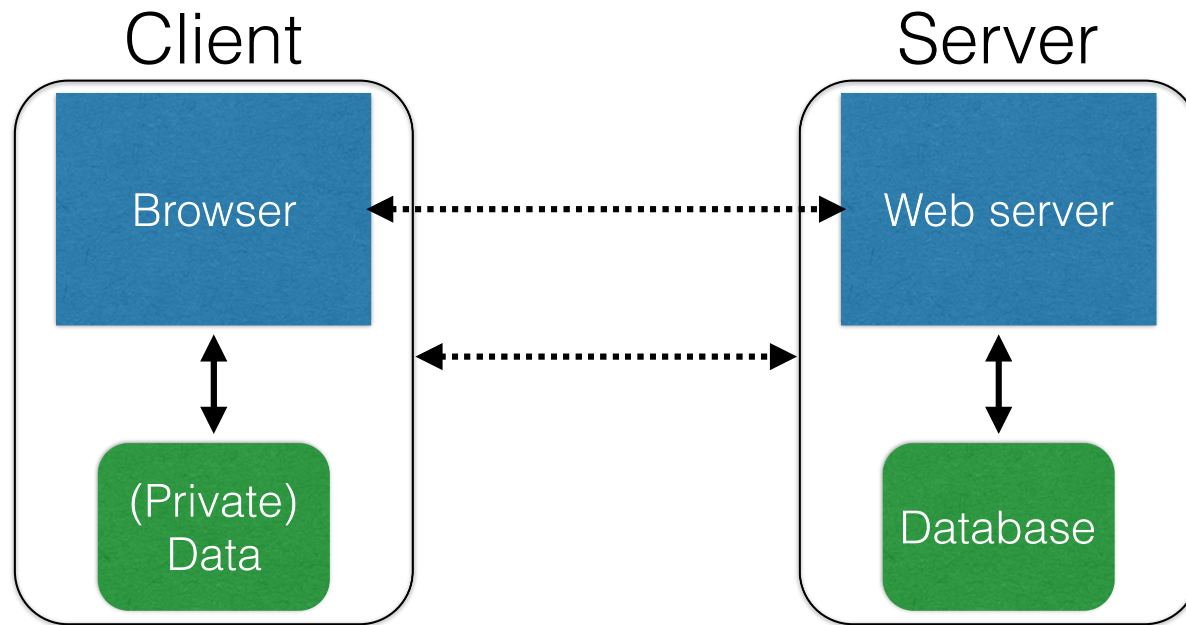
- Attack and defense at host machines
    - Applications written in C and C++
    - Violations of memory safety
- Web security now
    - Attacking web services
    - Problems: Confusion of code/data; untrusted input

Web security topics

- Web basics (today)
- SQL injection, defenses (today)
- Stateful web and session problems (Thursday)
- Dynamic web and XSS (Thursday)

Web Basics

The web, basically



(Much) user data is part of the browser

DB is a separate entity, logically (and often physically)

Interacting with web servers

Resources which are identified by a _URL_

(Universal Resource Locator)

`http://www.umiacs.umd.edu/~mmazurek/index.html`

Protocol

`ftp`
`https`
`tor`

Hostname/server

Translated to an IP address by DNS
(e.g., `128.8.127.3`)

Path to a resource

Here, the file `index.html` is static content
i.e., a fixed file returned by the server

Interacting with web servers

Resources* which are identified by a *URL

(Universal Resource Locator)

Path to a resource

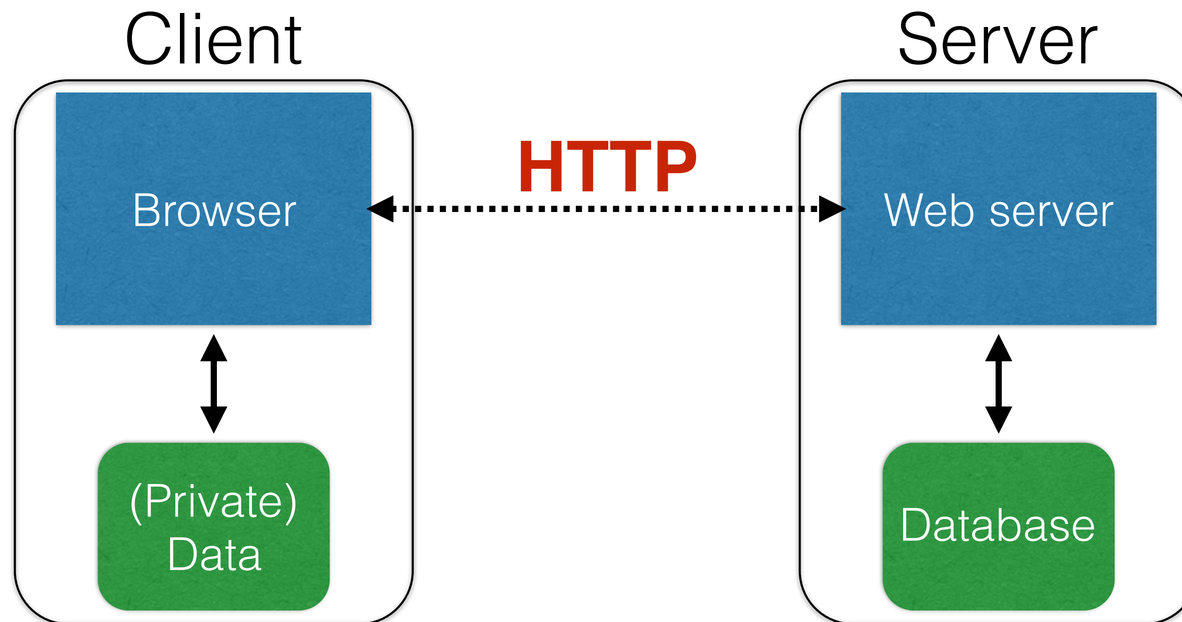
`http://facebook.com/delete.php?f=joe123&w=16`

Arguments

Here, the file **`delete.php`** is dynamic content

i.e., the server generates the content on the fly

Basic structure of web traffic



- HyperText Transfer Protocol (**HTTP**)
  - An "application-layer" protocol for exchanging data

Basic structure of web traffic

- Requests contain:
  - The **URL** of the resource the client wishes to obtain
  - **Headers** describing what the browser can do

- Request types can be **GET** or **POST**
  - **GET**: all data is in the URL itself
  - **POST**: includes the data as separate fields

HTTP GET requests

https://krebsonsecurity.com

HTTP Headers

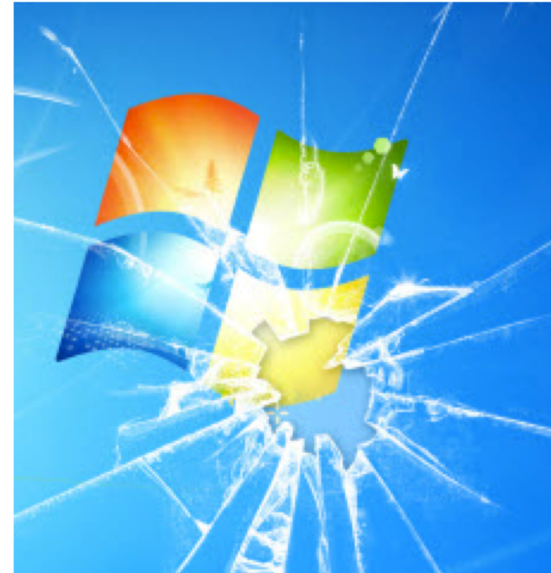
https://krebsonsecurity.com/

```
GET / HTTP/1.1
Host: krebsonsecurity.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:40.0) Gecko/20100101 Firefox/40.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
```

User-Agent is typically a **browser**
but it can be `wget`, JDK, etc.

According to security firm Shavlik, the patches that address flaws which have already been publicly disclosed include a large **Internet Explorer** (IE) update that corrects 17 flaws and a fix for **Microsoft Edge**, Redmond's flagship replacement browser for IE; both address this bug, among others.

A critical fix for a Windows graphics component addresses flaws that previously showed up in two public disclosures, one of which Shavlik says is currently being exploited in the wild (CVE-2015-2546). The 100th patch that Microsoft has issued so far this year — a salve for **Windows**



HTTP Headers

https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2015-1421

GET /view/vuln/detail?vulnId=CVE-2015-1421 HTTP/1.1
Host: web.nvd.nist.gov
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:40.0) Gecko/20100101 Firefox/40.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Referer: https://krebsonsecurity.com/
Connection: keep-alive

Referrer URL: site from which this request was issued.

HTTP POST requests

Posting on Piazza

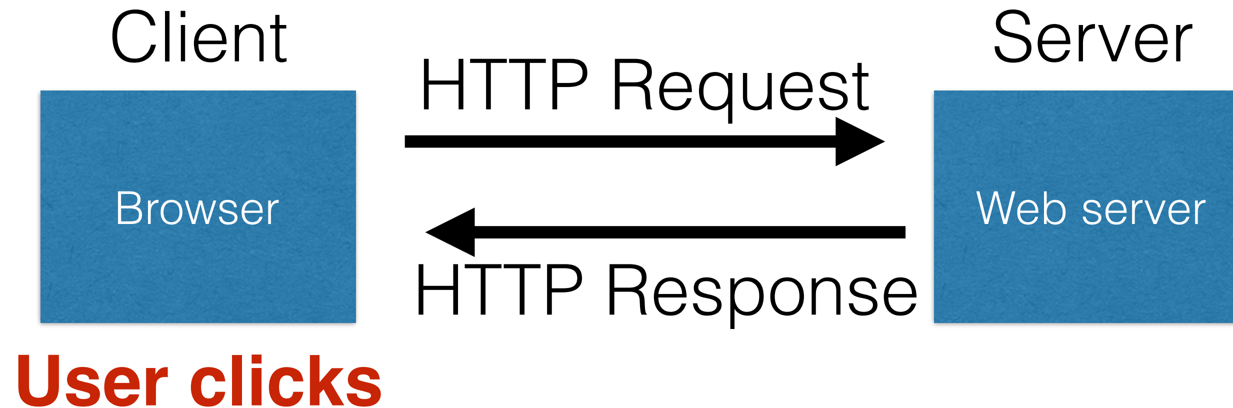
HTTP Headers

```
https://piazza.com/logic/api?method=content.create&aid=hrteve7t83et

POST /logic/api?method=content.create&aid=hrteve7t83et HTTP/1.1
Host: piazza.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.2.11) Gecko/20101013 Ubuntu/9.04 (jaunty) Firefox/3.6.11
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Referer: https://piazza.com/class
Content-Length: 339
Cookie: piazza_session="DFwuCEFIGvEGwwHLJyuCvHIGtHKECCKL.5%25x+x+ux%255M5%22%215%3F5%26x%26%26%7C%22%21r...
Pragma: no-cache
Cache-Control: no-cache
 {"method":"content.create","params":{"cid":"hrpng9q2nndos","subject":"<p>Interesting.. perhaps it has to do with a change to the ...
```

Implicitly includes data as a part of the URL

Explicitly includes data as a part of the request's content

Basic structure of web traffic



- **Responses** contain:
  - **Status** code
  - **Headers** describing what the server provides
  - **Data**
  - **Cookies** (much more on these later)
    - Represent *state* the server would like the browser to store

HTTP responses

HTTP version

Status code

Reason

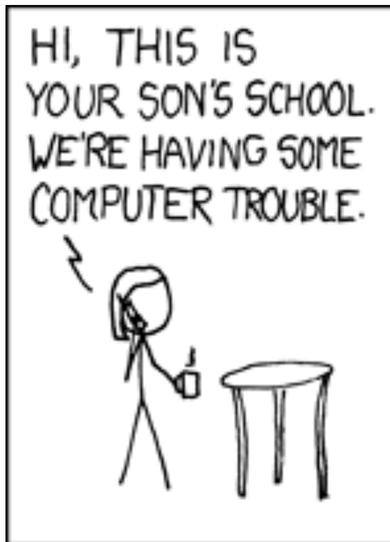
Headers

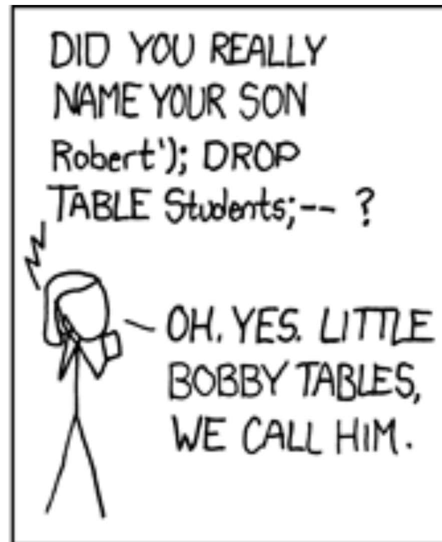
```
HTTP/1.1 200 OK
Cache-Control: private, no-store, must-revalidate
Content-Length: 50567
Content-Type: text/html; charset=utf-8
Server: Microsoft-IIS/7.5
Set-Cookie: CMSPreferredCulture=en-US; path=/; HttpOnly; Secure
Set-Cookie: ASP.NET_SessionId=4l2oj4nthxmvjs1waletxlqa; path=/; secure; HttpOnly
Set-Cookie: CMSCurrentTheme=NVDLegacy; path=/; HttpOnly; Secure
X-Frame-Options: SAMEORIGIN
x-ua-compatible: IE=Edge
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET, ASP.NET
```

Data

```
<html> ...... </html>
```

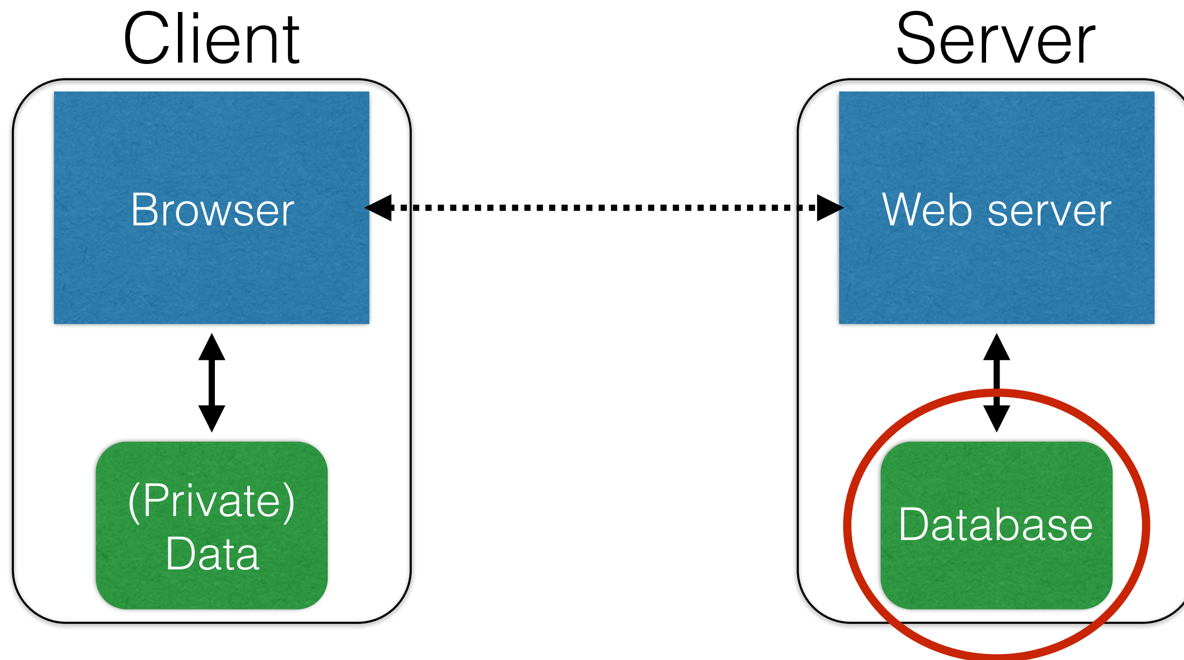
SQL injection





http://xkcd.com/327/

Server-side data



Long-lived state, stored in a separate *database*

Need to **protect this state** from illicit access and tampering

Databases

- Provide data storage & manipulation
- Database designer organizes data into tables
- Programmers query the database
- Database Management Systems (DBMSes) provide
    - semantics for how to organize data
    - transactions for manipulating data sanely
    - a **language** for creating & querying data
        - and APIs to interoperate with other languages
    - management via users & permissions

SQL (Standard Query Language)

Table

Users

Table name

Name	Gender	Age	Email	Password
Connie	F	12	connie@bc.com	sw0rdg1rl
Steven	M	14	steven@bc.com	c00kieC4t
Greg	M	34	mr.uni@bc.com	i<3ros3!
Vidalia	M	35	vidalia@bc.com	sc&On!0N

Row (Record)

Column

```
SELECT Age FROM Users WHERE Name='Greg'; 34

UPDATE Users SET email='mr.uni@bc.com'
   WHERE Age=34; -- this is a comment

INSERT INTO Users Values('Pearl', 'F', ...);
DROP TABLE Users;
```

Server-side code

Website

“Login code” (PHP)

```
$result = mysql_query(“select * from Users
      where(name=‘$user’ and password=‘$pass’);”);
```

Suppose you successfully log in as $user
if this returns any results

How could you exploit this?

SQL injection

frank' OR 1=1); --

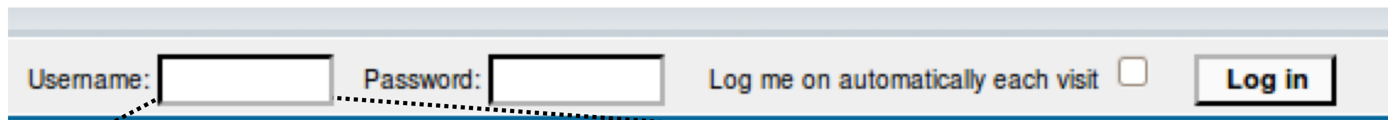
```
$result = mysql_query("select * from Users
        where(name='$user' and password='$pass');");
```

```
$result = mysql_query("select * from Users
        where(name='frank' OR 1=1); --
            and password='whocares');");
```

Login successful!

Problem: Data and code mixed up together

SQL injection: Worse



frank’ OR 1=1); DROP TABLE Users; --

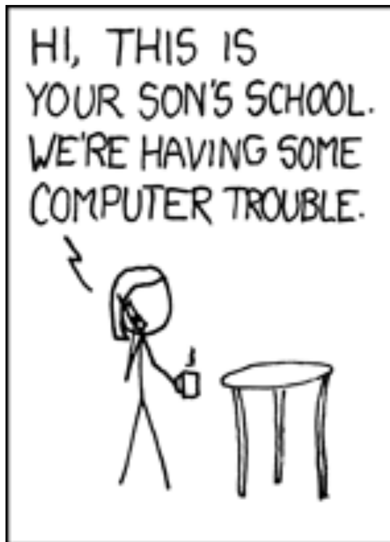
```
$result = mysql_query(“select * from Users
        where(name=‘$user’ and password=‘$pass’);”);

$result = mysql_query(“select * from Users
        where(name=‘frank’ OR 1=1);
        DROP TABLE Users; --
            and password=‘whocares’);”);
```

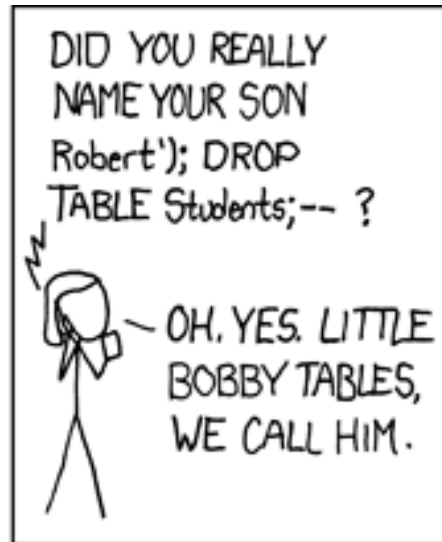
Can chain together statements with semicolon:
STATEMENT 1 ; STATEMENT 2

SQL injection: Even worse

```
'); EXEC cmdshell 'net user badguy backdoor / ADD'; --
```

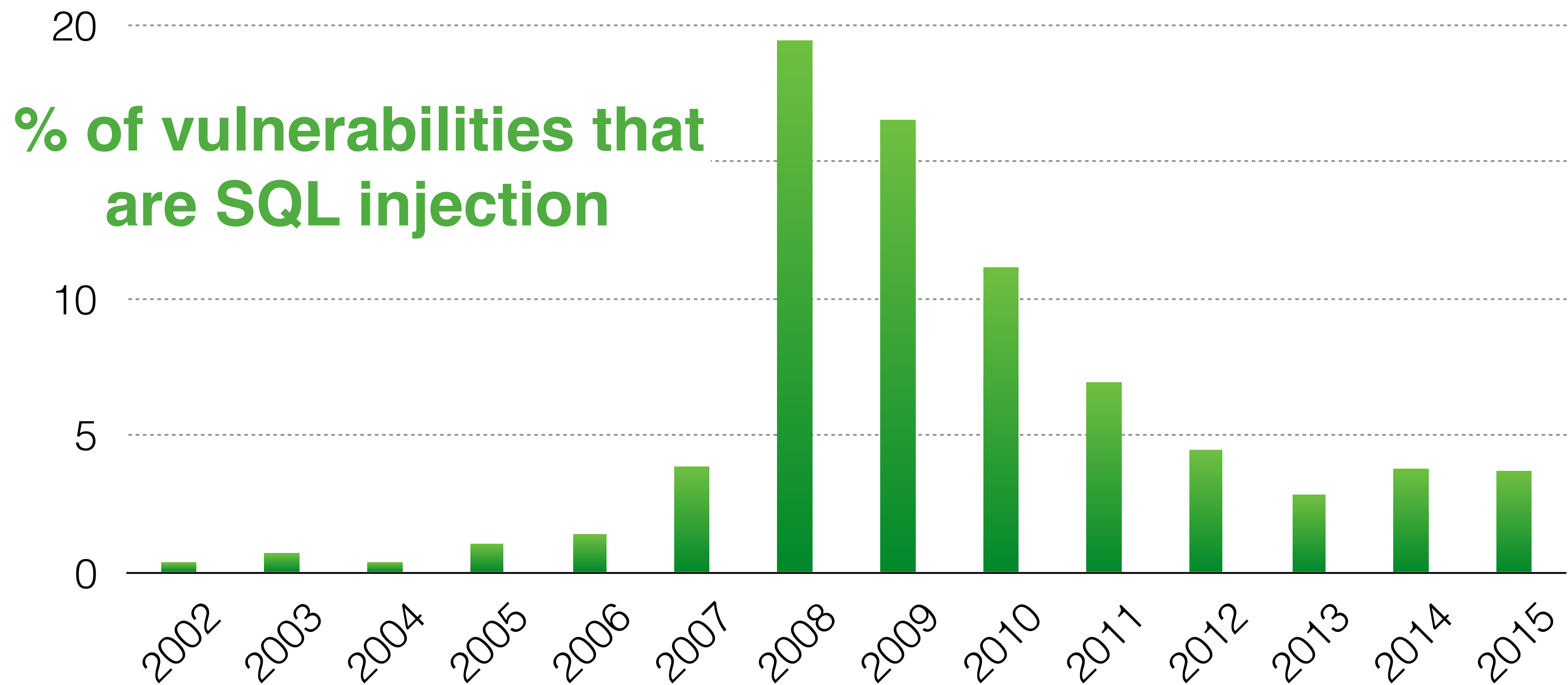
```
$result = mysql_query("select * from Users
        where(name='$user' and password='$pass');");

$result = mysql_query("select * from Users
        where(name='');
        EXEC cmdshell 'net user badguy backdoor / ADD'; --
            and password='whocares');");
```





http://xkcd.com/327/

SQL injection attacks are common



http://web.nvd.nist.gov/view/vuln/statistics

ZU 0666',0,0); DROP DATABASE TABLICE;

SQL injection countermeasures

The underlying issue

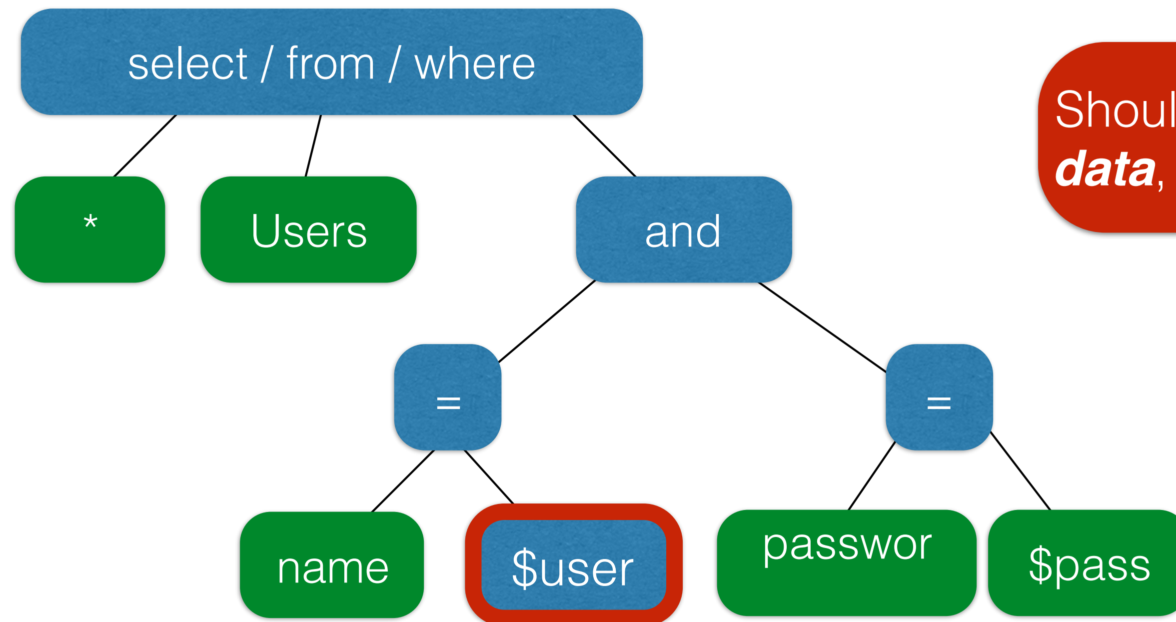
```
$result = mysql_query("select * from Users
        where(name='$user' and password='$pass');");
```

- This one string combines the **code** and the **data**
  - Similar to buffer overflows

When the boundary between code and data blurs, we open ourselves up to vulnerabilities

The underlying issue

```
$result = mysql_query(“select * from Users
        where(name=‘$user’ and password=‘$pass’);”);
```



Should be ***data***, not ***code***

Prevention: Input validation

- We require input of a certain form, but we cannot guarantee it has that form, so we must **validate it**
  - Just like we do to avoid buffer overflows
- Making input trustworthy
  - **Check** it has the expected form, reject it if not
  - **Sanitize** by modifying it or using it such that the result is correctly formed

Sanitization: Blacklisting

`'  ;  --`

- **Delete** the characters you don't want
- **Downside**: "`Lupita Nyong'o`"
  - You want these characters sometimes!
  - How do you know if/when the characters are bad?
- **Downside**: How to know you've ID'd all bad chars?

Sanitization: Escaping

- **Replace** problematic characters with safe ones
  - Change `'` to `\'`
  - Change `;` to `\;`
  - Change `-` to `\-`
  - Change `\` to `\\`
- Hard by hand, there are many libs & methods
  - `magic_quotes_gpc = On`
  - `mysql_real_escape_string()`
- **Downside**: Sometimes you want these in your SQL!
  - And escaping still may not be enough

Checking: Whitelisting

- Check that the user input is **known to be safe**
  - E.g., integer within the right range
- Rationale: Given invalid input, **safer to reject than fix**
  - “Fixes” may result in wrong output, or vulnerabilities
  - Principle of fail-safe defaults
- **Downside**: Hard for rich input!
  - How to whitelist usernames? First names?

Sanitization via escaping, whitelisting, blacklisting is HARD.

Can we do better?

Sanitization: Prepared statements

- Treat user data according to its *type*
  - Decouple the code and the data

```
$result = mysql_query(“select * from Users
       where(name=‘$user’ and password=‘$pass’);”);
```

```
$db = new mysql(“localhost”, “user”, “pass”, “DB”);

$statement = $db->prepare(“select * from Users
       where(name=? and password=?);”);
```

Bind variables

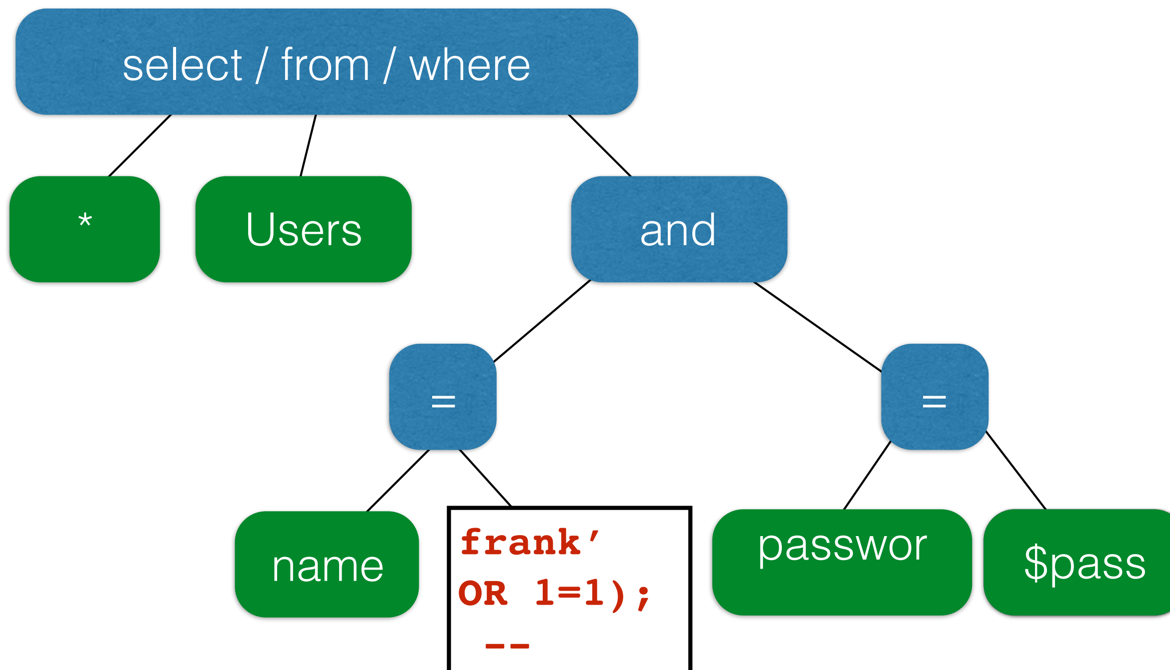
```
$statement->bind_param(“ss”, $user, $pass);
$statement->execute();
```

Bind variables are typed

Decoupling lets us compile now, before binding the data

Using prepared statements

```
$statement = $db->prepare("select * from Users
       where(name=?       and password=?);");
$stmt->bind_param("ss", $user, $pass);
```



Binding is only applied to the leaves, so the structure of the tree is *fixed*

Additional mitigation

- For **defense in depth**, *also* try to mitigate any attack
  - But should **always do input validation** in any case!
- **Limit privileges**; reduces power of exploitation
  - Limit commands and/or tables a user can access
  - e.g., allow SELECT on Orders but not Creditcards
- **Encrypt sensitive data**; less useful if stolen
  - May not need to encrypt Orders table
  - But certainly encrypt creditcards.cc_numbers