How to SQLi

- Network Security
  - Firewalls
  - Intrusion Detection Systems
  - Physical access
  - Attached devices

- (Web) Application Security
  - Secure code development (of all components)

- Hardware / Embedded Systems / SCADA & ICS Security
  - Physical access
  - Network access
  - Secure code (firmware) development

- Physical Security
  - Social Engineering
  - Operational security (situational awareness)

- Policy & Compliance

- Forensics
- Malware
- Exploit Development

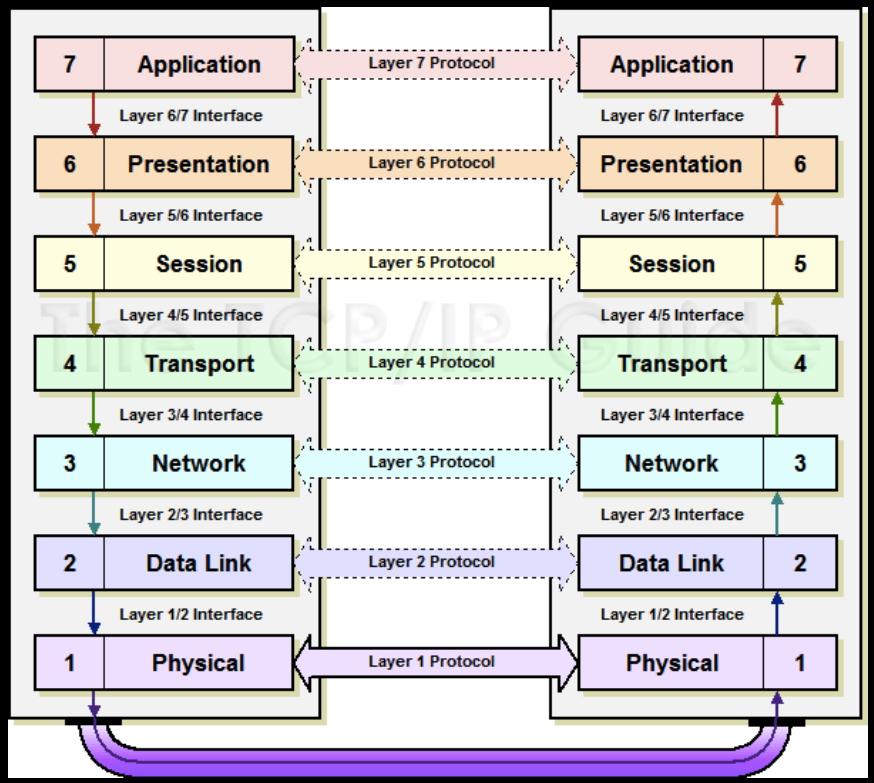
Tips

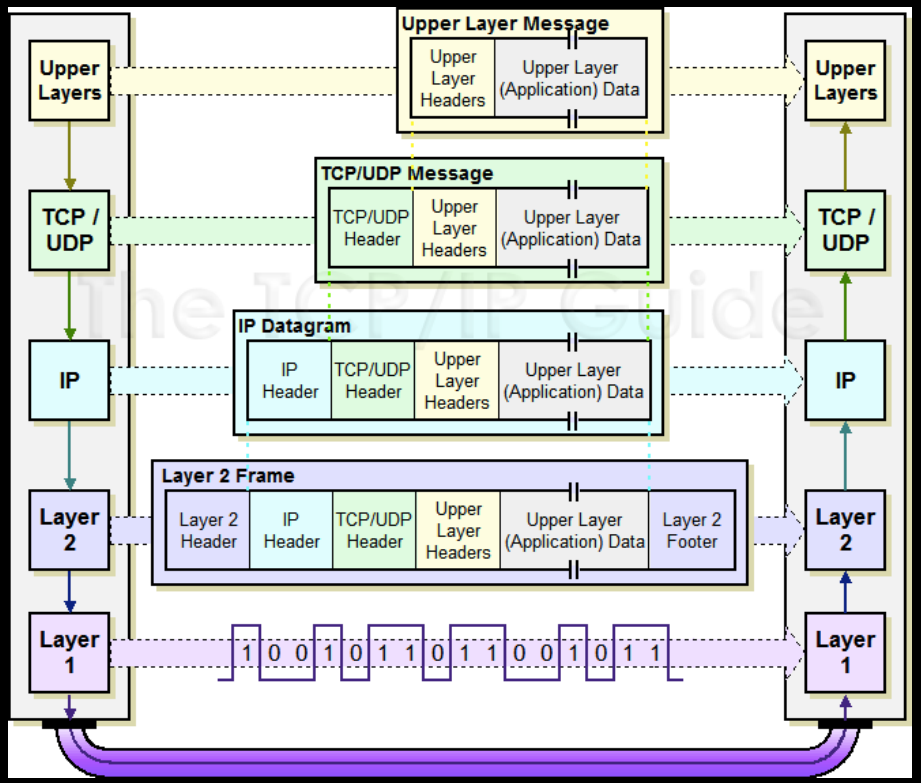
- Stay away from automated tools (for stealth / education)

- Use the application as it is meant to be used
- Observe how data moves
- Send bad input, misuse the application

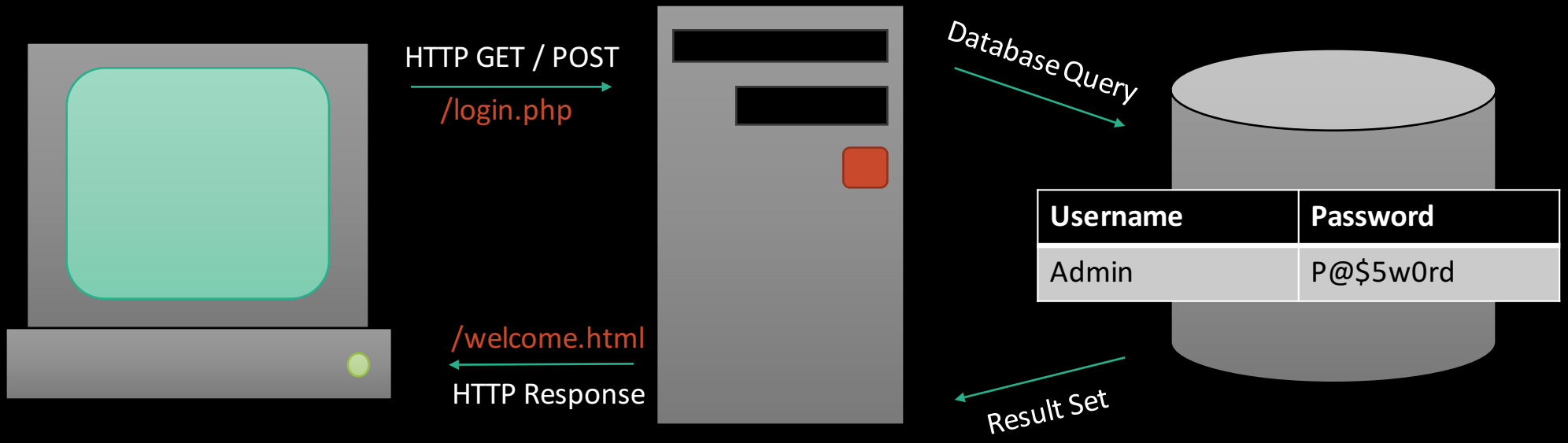
- Practice makes perfect

At the Network Level





Application <-> Database Interaction



<Insert plug about Burpsuite here>

Anatomy

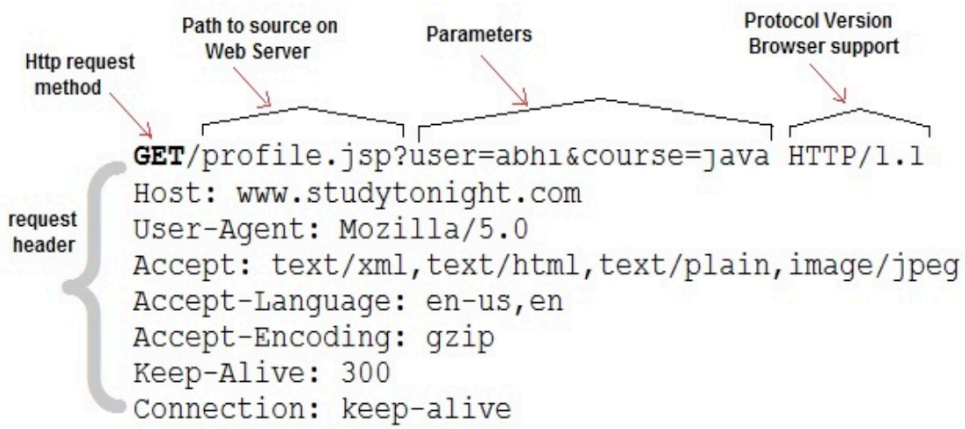
- Frontend
  - Content delivered to client
  - HTML, JavaScript, CSS, Ajax
- Middle layer
  - PHP, Python, Node, ASP.NET
  - Pages are dynamically generated
  - Request parameters are parsed
- Database
  - MySQL, MSSQL, PostgreSQL, Sqlite
  - Direct queries or through ORM

Important Things

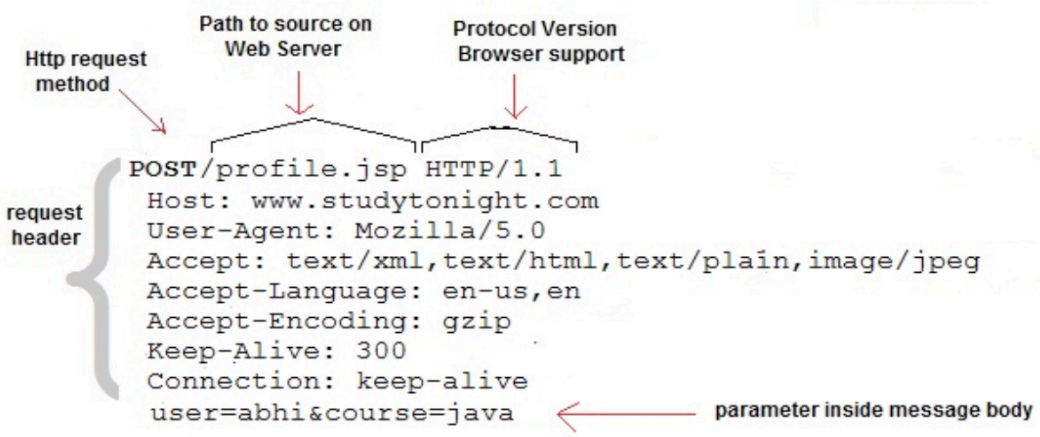
- HTTP Request / Response
- Difference between GET and POST requests?

Method	Description
GET	Request to read a Web page
HEAD	Request to read a Web page's header
PUT	Request to store a Web page
POST	Append to a named resource (e.g., a Web page)
DELETE	Remove the Web page
TRACE	Echo the incoming request
CONNECT	Reserved for future use
OPTIONS	Query certain options

GET Request



POST Request



Important Things ++

- Cookies
  - Data sent by the web server that is stored in the browser and sent back in subsequent requests to maintain state / record browsing activity (Session management, storage, sort of an ID badge)
- Cookie Attributes
  - Domain and Path
    - Defines scope of cookie
  - Expires and Max-age
    - Defines when the browser should delete the cookie
  - Secure
    - Directs the browser on whether or not to send the cookie over encrypted connection only or not
  - HttpOnly
    - Directs the browser on JavaScript's access to the cookie

```
GET /index.html HTTP/1.1
Host: www.example.org
...

HTTP/1.0 200 OK
Content-type: text/html
Set-Cookie: theme=light
Set-Cookie: sessionToken=abc123;
Expires=Wed, 09 Jun 2021 10:18:14 GMT
...

GET /spec.html HTTP/1.1
Host: www.example.org
Cookie: theme=light; sessionToken=abc123
...
```

(Web) Application Vulnerabilities

- Injection (SQL, JavaScript, etc.)
- Broken Authentication and Session Management
- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF)
- Sensitive Data Exposure
- Local File Inclusion
- Business Logic Attacks

Check Yo Inputs

- Users can interfere with data transmitted between the client and the server (parameters, cookies, HTTP headers…) Any client-side controls can be easily circumvented
- Users can send requests out of sequence, and submit parameters in another order than what the application expects, numerous times, or not at all. Devs make assumptions about user interactions, this creates vulns
- A browser is not the only way to access a web app, numerous tools exist that can operate with or without a web browser to attack an application

Check Yo Self

- SSL encrypts data, protecting it from being viewed or modified in transit
- SSL does not stop an attacker from submitting crafted inputs to the server
- Writing your own security mechanisms is a BAD IDEA
- Authentication, Session management, and Access controls are separate attack surfaces
- Whitelisting inputs > blacklisting inputs
  - data can be encoded
  - you'll probably forget something
    - `SELECT` is blocked, try `SeLeCt`
    - `or 1=1--` is blocked, try `or 2=2--`
    - `alert('xss')` is blocked, try `prompt('xss')`

How to SQL

```
SELECT something FROM table WHERE field =
'value';
```

- Disregarding optimization…
- Database looks at each row in the table
- If the column values in the row match what's being requested in the WHERE clause (statement evaluates to TRUE), the data of the requested field (column) is added to the result set
- The database finishes parsing the table and returns the result set

SQL statement that uses user input

```
SELECT username, password FROM users WHERE
username = '" + username + "' AND password = '"
+ password + "';

SELECT username, password FROM users WHERE
username = 'Admin' AND password =
'Lamepassword';
```

What is SQL (or any code) Injection?

Code v. data problem

User input is interpreted as code and executed

```
SELECT username, password FROM users WHERE
username = 'Admin' AND password = ''or'1'='1';
```

All rows will be returned in the result set because 1 will always equal 1, this statement will evaluate to TRUE for every row

What can we do with this?

- Bypass login
- Exfiltrate data
- Elevate privilege (admin creds!)
- Tamper with logs and records
- Delete everything

- **If SQLi is found in a web application, it should automatically be considered an critical vulnerability**

How to find it

- Supply unexpected input such as ‘ ”) --
- Identify any error messages or changes in response / behavior
- Determine if your input is being executed as code
- Types of SQLi
  - Regular: is extra data returned?
  - Equivalency: are statements executed differently?
  - Blind: see if you can cause a backend delay or out-of-band response

Testing

- Does the DB send an error when it receives ’ or ” or) or --
- What is the error? Is the statement included?
- Does sending ‘ ‘ (two single ticks) alleviate the error?
- Test to see if the DB does the same thing when you input FOO as it does when you input:
- ‘||’FOO (Oracle)
- ‘+’FOO (MS-SQL)
- ‘ ‘FOO (space between the single ticks) (MySQL)

Numerical Testing

- Try math
  - 1+1
  - 3-1
- Or functions
  - 67-ASCII(‘A’)
  - 51-ASCII(1)
- These should evaluate to 2 if being processed by the DB

Watch out for encoding

Certain SQL characters are also special characters in HTTP

&	%26
=	%3d
(space)	%20
+	%2b
;	%3b

Fingerprinting the Database

- Issue DB specific commands

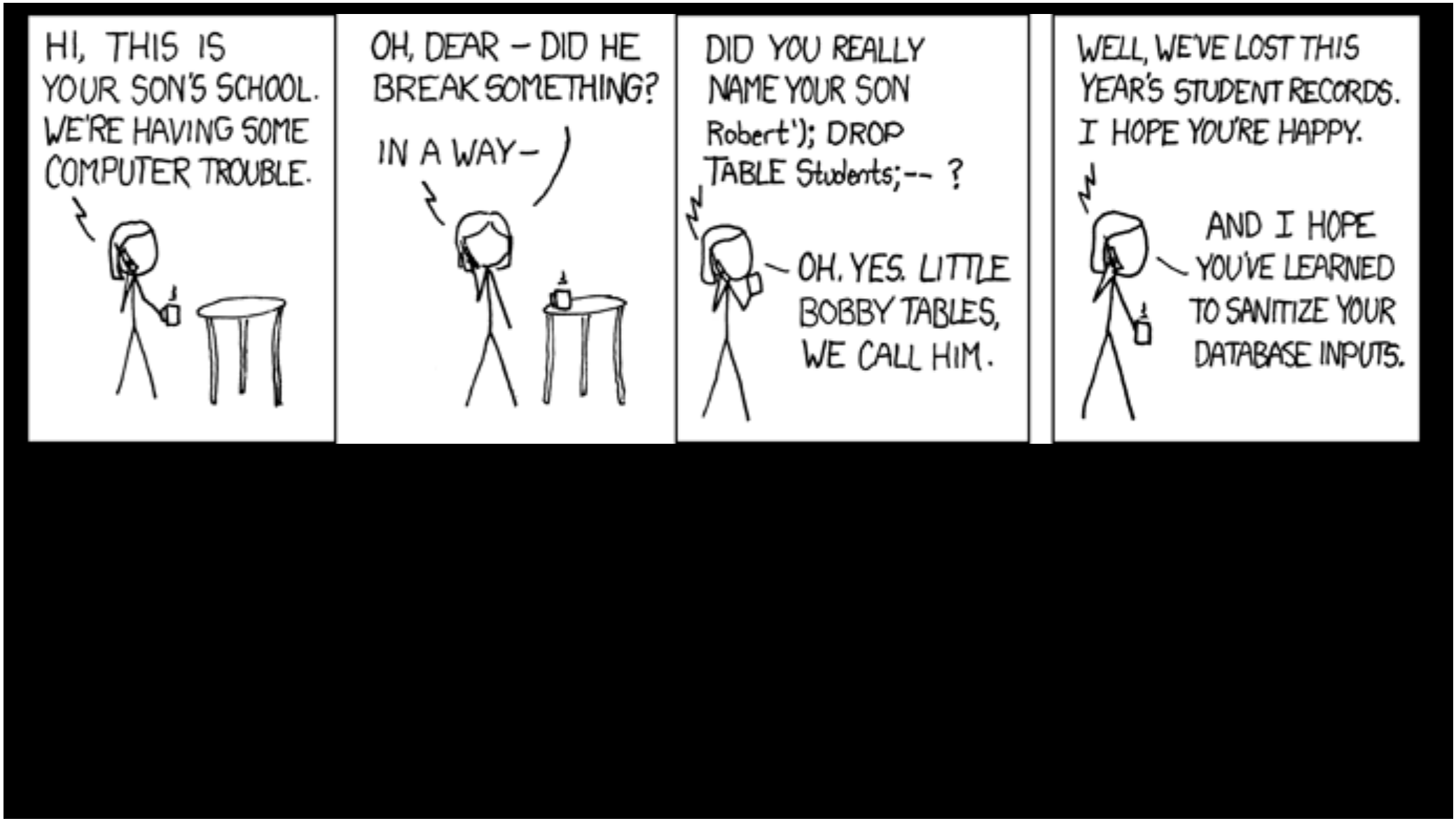
	Text Data	Numeric data	Blind
Oracle	’foo’||’bar’	BITAND(1,1)-BITAND(1,1)	
MS-SQL	’foo’+’bar’	@@PACK_RECIEVED-@@PACK_RECEIVED	a’ WAITFOR DELAY ‘00:00:05
MySQL	’foo’ ‘bar’	CONNECTION_ID()-CONNECTION_ID()	a’ sleep(5000)

- Sometimes you need to comment out the rest of the statement

Oracle	-- or /*
MS-SQL	--
MySQL	# or /*
SQLite	-- or /*
PostgreSQL	--

Mitigation

- Parameterized Queries (prepared statements)
  - Define SQL code first, pass in parameters later
  - DB can distinguish between code and data, regardless of input
  - The attacker cannot change the query, even by inserting SQL commands


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.

Wrong

```
“SELECT username, password FROM web30 WHERE username = ‘” +
username + “’ AND password = ‘” + password + “’;”
```

Exploit

```
“SELECT username, password FROM web30 WHERE username = ‘test’
AND password = ‘‘or’1’=‘1’;”
```

or

```
“SELECT username, password FROM web30 WHERE username =
‘’or’1’=‘1’--
```

Right

```
“SELECT username, password FROM web30 WHERE username = %s AND
password = %s;”
```

More resources

- The Open Web Application Security Project Top 10 :
  - https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
- Guide to Preventing SQLi:
  - http://bobby-tables.com/
- The Web Application Hacker's Handbook (pdf):
  - https://leaksource.files.wordpress.com/2014/08/the-web-application-hackers-handbook.pdf
- Tangled Web (pdf):
  - http://venom630.free.fr/pdf/The%20Tagled%20Web%20A%20Guide%20to%20Securing%20Modern%20Web%20Applications.pdf
- https://www.owasp.org/index.php/Query_Parameterization_Cheat_Sheet
- http://blog.codinghorror.com/give-me-parameterized-sql-or-give-me-death/
- http://pentestmonkey.net/category/cheat- sheet/sql-injection
- http://www.unixwiz.net/techtips/sql- injection.html

partial content from Andrew McKenna, who taught i415 SP 2015