CSE 484 : Computer Security and Privacy

Web Security
[Web Application Security]

Winter 2021

David Kohlbrenner

dkohlbre@cs.washington.edu

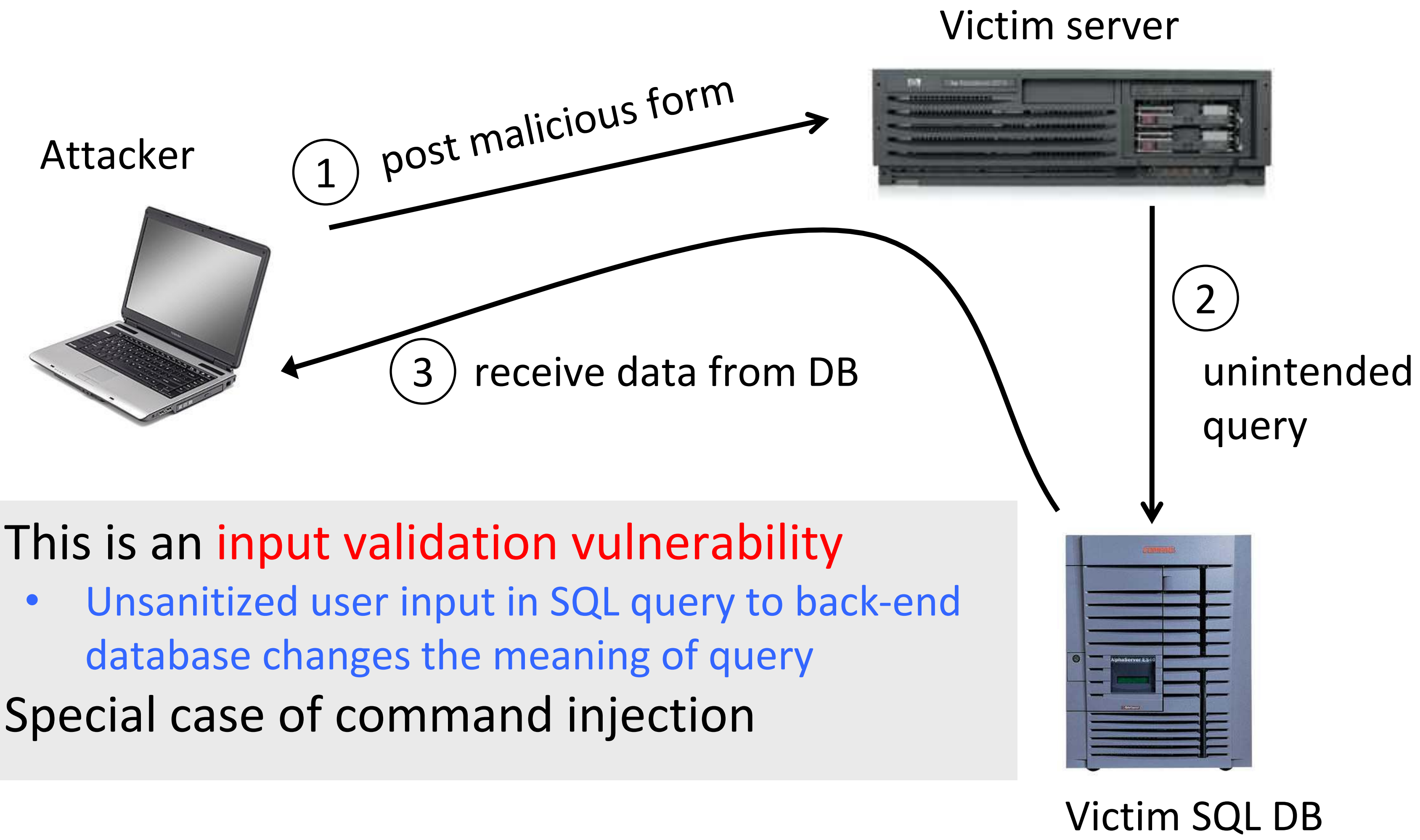
Thanks to Franzi Roesner, Dan Boneh, Dieter Gollmann, Dan Halperin, Yoshi Kohno, Ada Lerner, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

Admin

- Lab 2
  - Granting access on a regular basis
  - Please sign up if you haven’t already
- Final project
  - First checkpoint deadline TODAY!

SQL Injection

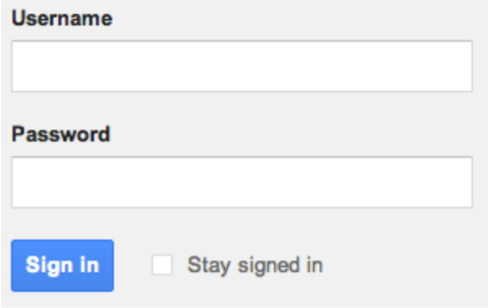
SQL Injection: Basic Idea



- 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

Authentication with Backend DB

```
set UserFound = execute(
     “SELECT * FROM UserTable WHERE
     username= ‘ ”  &  form(“user”)  & “ ’ AND
     password=  ‘ ”  &  form(“pwd”)  & “ ’ ” );
```



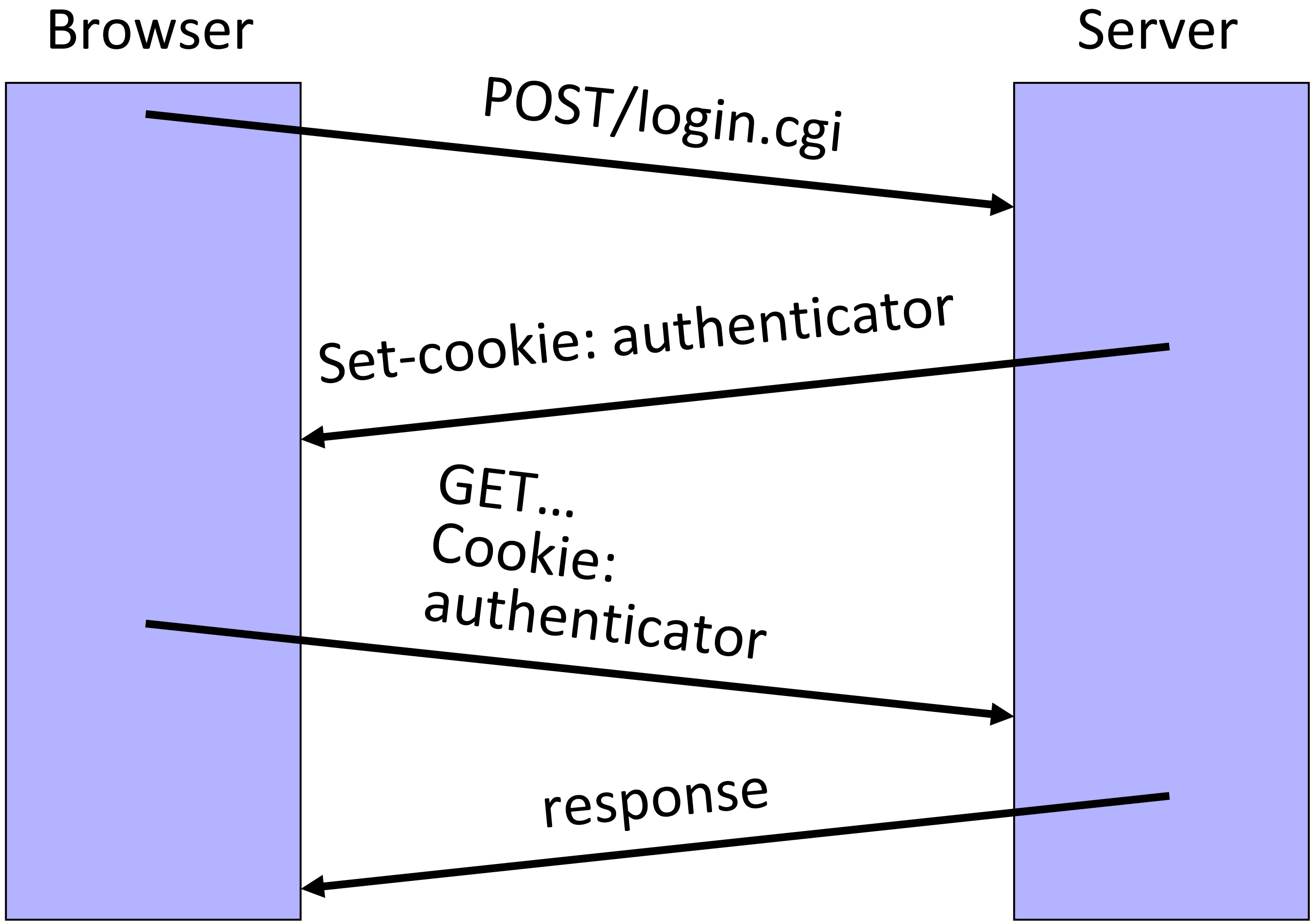
User supplies username and password, this SQL query checks if user/password combination is in the database

```
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

Cross-Site Request Forgery (CSRF/XSRF)

Cookie-Based Authentication Redux



Browser Sandbox Redux

- Based on the same origin policy (SOP)
- Active content (scripts) can send anywhere!
    - For example, can submit a POST request
    - Some ports inaccessible -- e.g., SMTP (email)
- Can only *read* response from the *same origin*
    - … but you can do a lot with just sending!

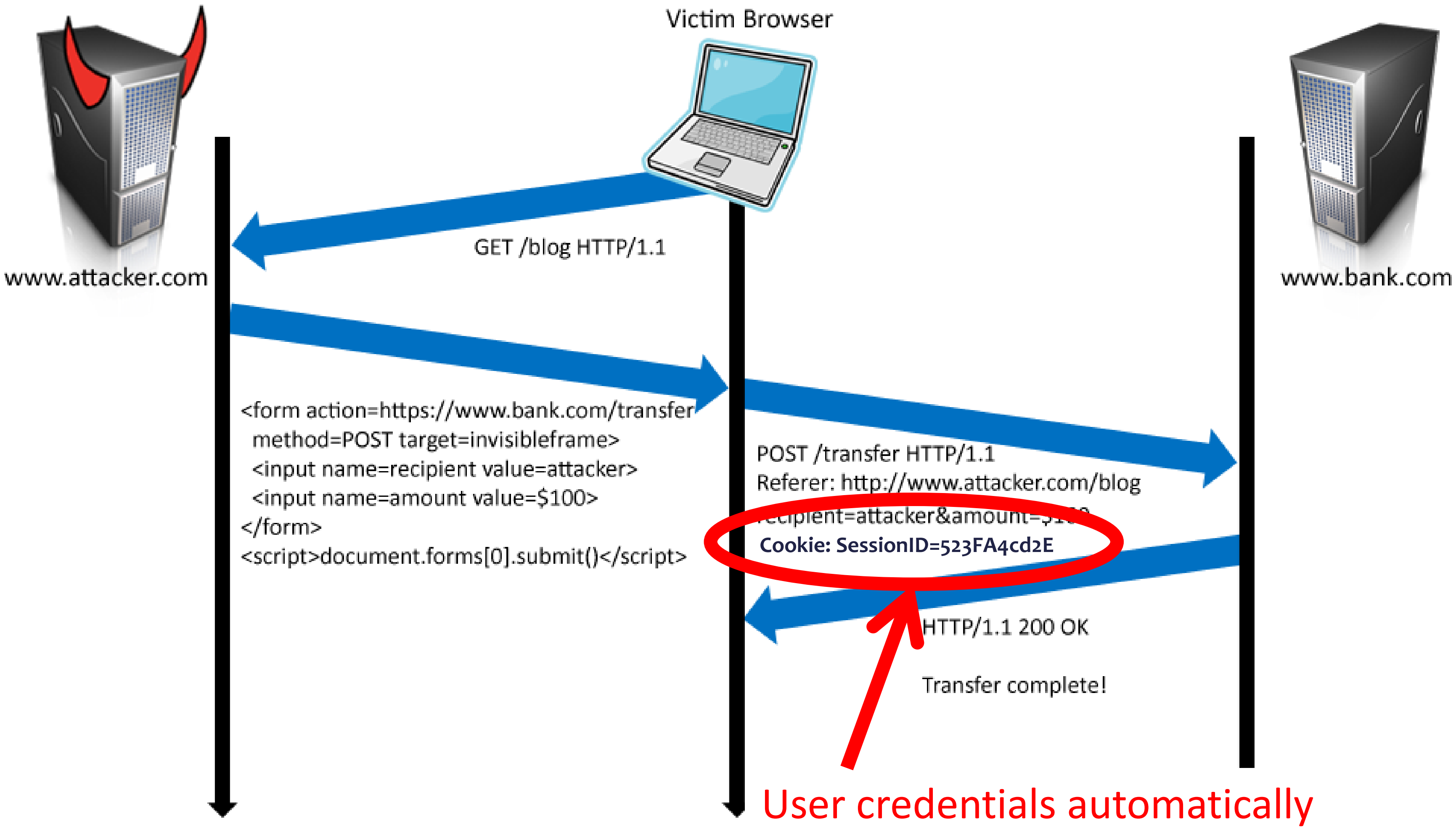
Cross-Site Request Forgery

- Users logs into bank.com, forgets to sign off
  - Session cookie remains in browser state
- User then visits a malicious website containing

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

- Browser sends cookie, payment request fulfilled!
- Lesson: cookie authentication is not sufficient when side effects can happen

Cookies in Forged Requests



```
<form action=https://www.bank.com/transfer
  method=POST target=invisibleframe>
  <input name=recipient value=attacker>
  <input name=amount value=$100>
</form>
<script>document.forms[0].submit()</script>
```

```
POST /transfer HTTP/1.1
Referer: http://www.attacker.com/blog
recipient=attacker&amount=$100
Cookie: SessionID=523FA4cd2E
```

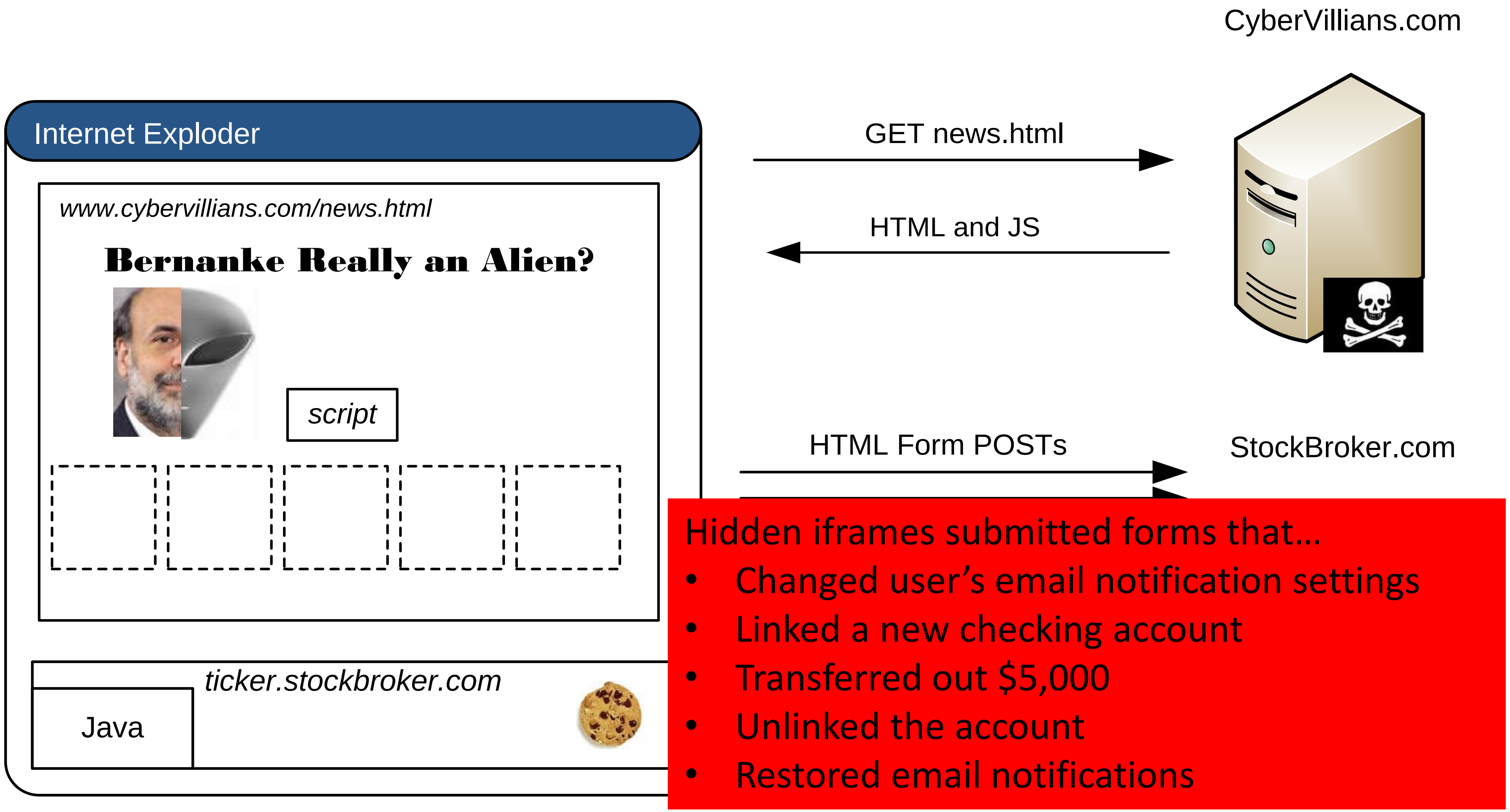
User credentials automatically sent by browser

Impact

- Hijack any ongoing session (if no protection)
  - Netflix: change account settings, Gmail: steal contacts, Amazon: one-click purchase
- Reprogram the user’s home router
- Login to the *attacker’s* account
  - Why?

XSRF True Story

[Alex Stamos]



CyberVillians.com

Internet Exploder

www.cybervillians.com/news.html

Bernanke Really an Alien?

script

GET news.html

HTML and JS

HTML Form POSTs

StockBroker.com

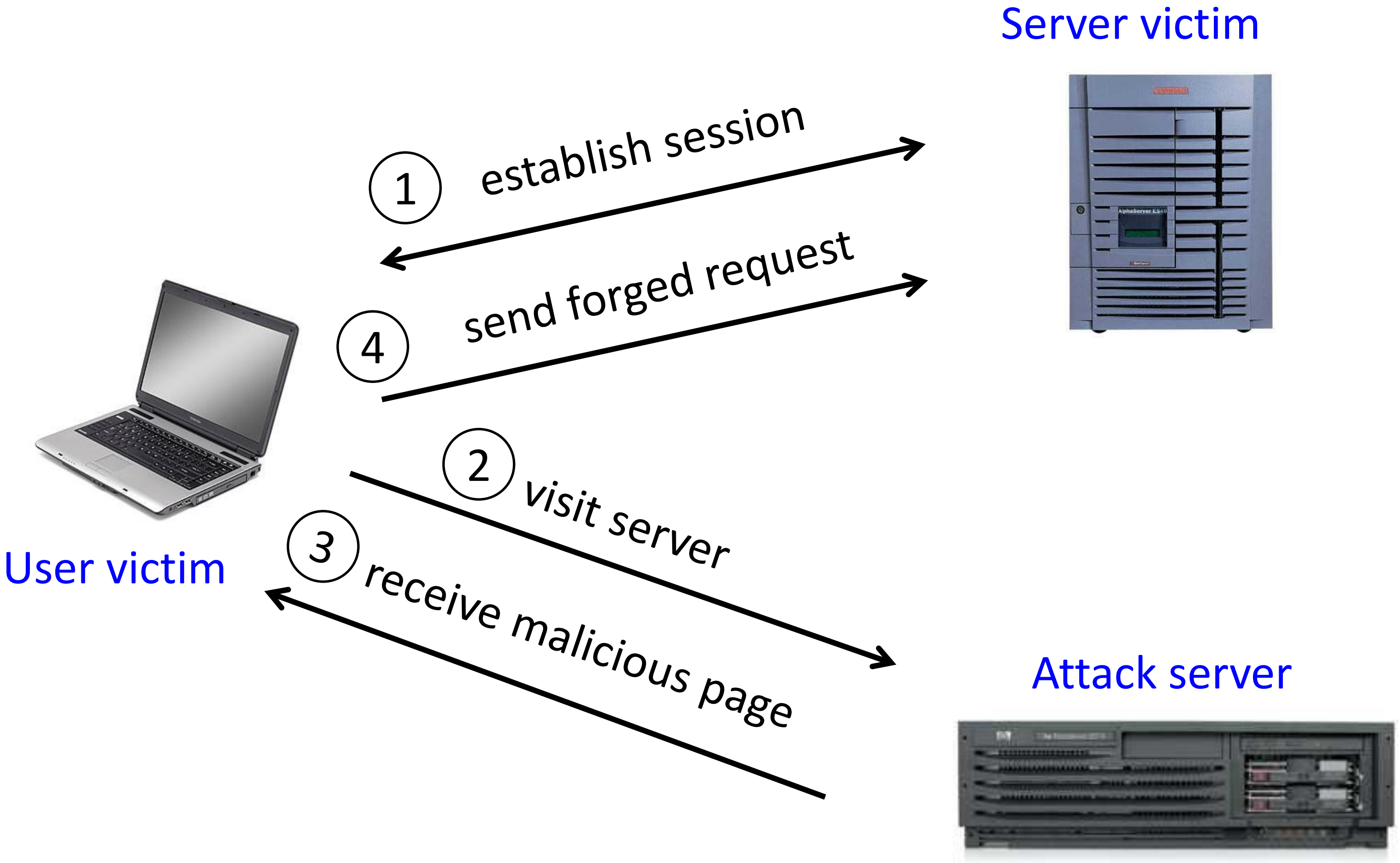
ticker.stockbroker.com

Java

Hidden iframes submitted forms that…

- Changed user’s email notification settings
- Linked a new checking account
- Transferred out $5,000
- Unlinked the account
- Restored email notifications

XSRF (aka CSRF): Summary



Server victim

1 establish session

4 send forged request

User victim

2 visit server

3 receive malicious page

Attack server

Q: how long do you stay logged on to Gmail? Financial sites?

Broader View of XSRF

- Abuse of cross-site data export
  - SOP does not control data export
  - Malicious webpage can initiates requests from the user's browser to an honest server
  - Server thinks requests are part of the established session between the browser and the server (automatically sends cookies)

XSRF Defenses

- Secret validation token

```
<input type=hidden value=23a3af01b>
```

- Referer validation

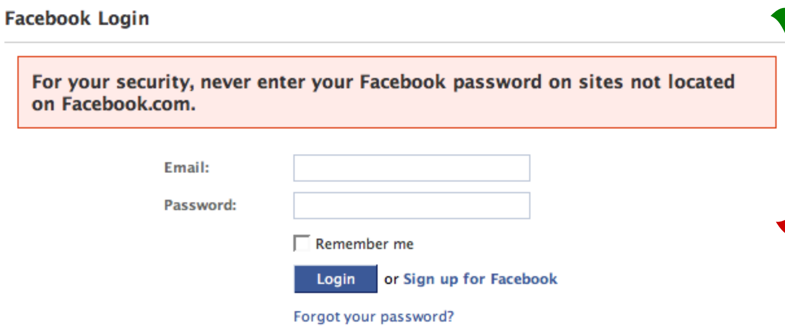
```
Referer:
http://www.facebook.com/home.php
```

Add Secret Token to Forms

```
<input type=hidden value=23a3af01b>
```

- "Synchronizer Token Pattern"
- Include a secret challenge token as a hidden input in forms
  - Token often based on user's session ID
  - Server must verify correctness of token before executing sensitive operations
- Why does this work?
  - **Same-origin policy:** attacker can't read token out of legitimate forms loaded in user's browser, so can't create fake forms with correct token

Referer Validation


Facebook Login

For your security, never enter your Facebook password on sites not located on Facebook.com.

Email:

Password:

□ Remember me

✓ Referer: http://www.facebook.com/home.php

✗ Referer: http://www.evil.com/attack.html

? Referer:

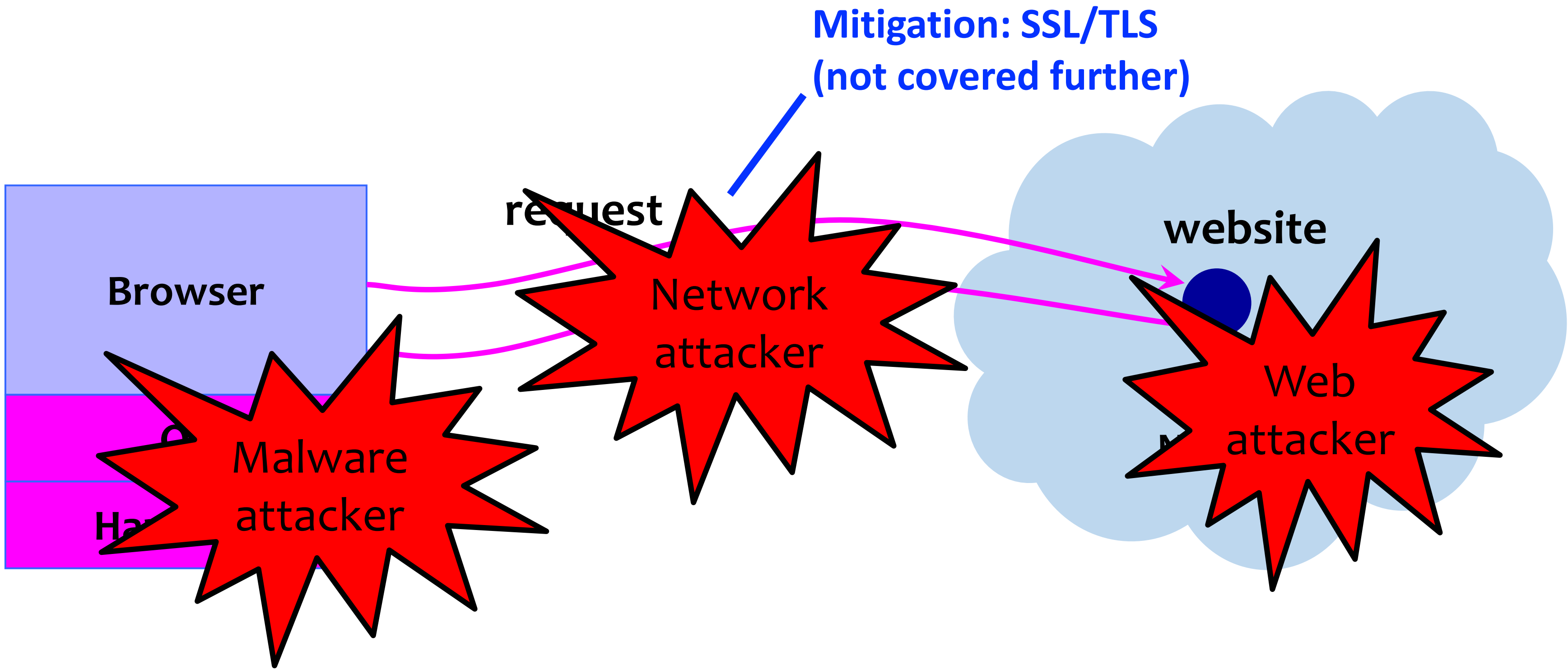
- Lenient referer checking – header is optional
- Strict referer checking – header is required

Why Not Always Strict Checking?

- Why might the referer header be suppressed?
  - Stripped by the organization’s network filter
  - Stripped by the local machine
  - Stripped by the browser for HTTPS → HTTP transitions
  - User preference in browser
  - Buggy browser
- Web applications can’t afford to block these users
- Many web application frameworks include CSRF defenses today

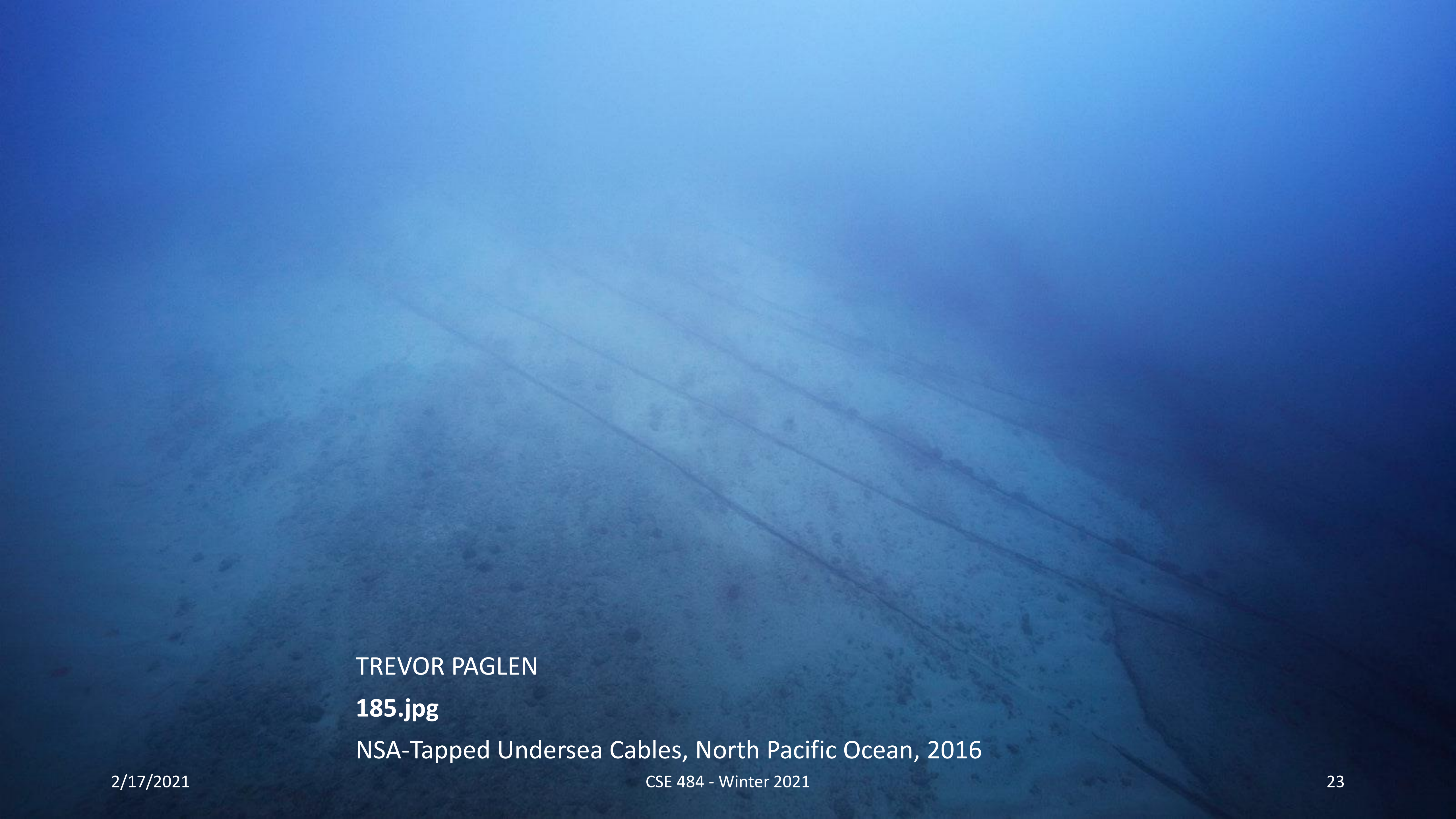
Bonus topic:
Consider the network

Where Does the Attacker Live?



Network attacker

- Lives between you and your destination server
  - Person-in-the-middle
  - Person-on-the-side
  - Passive/active
  - Physical/remote



TREVOR PAGLEN

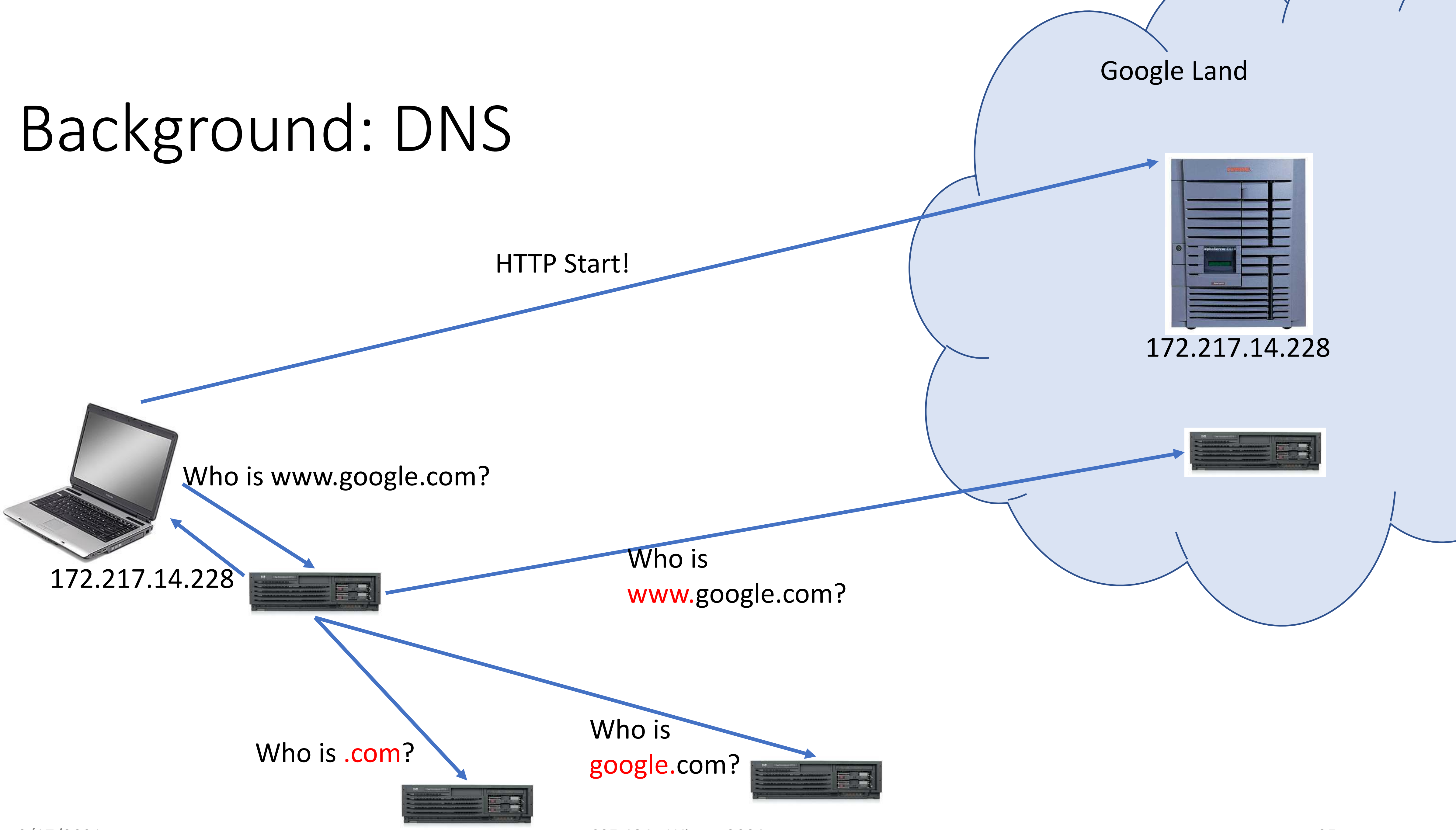
185.jpg

NSA-Tapped Undersea Cables, North Pacific Ocean, 2016

What might they be interested in?

- Eavesdropping
- Making us talk to the wrong server
- Denial-of-service
- Corrupting our conversation with a real server

Background: DNS

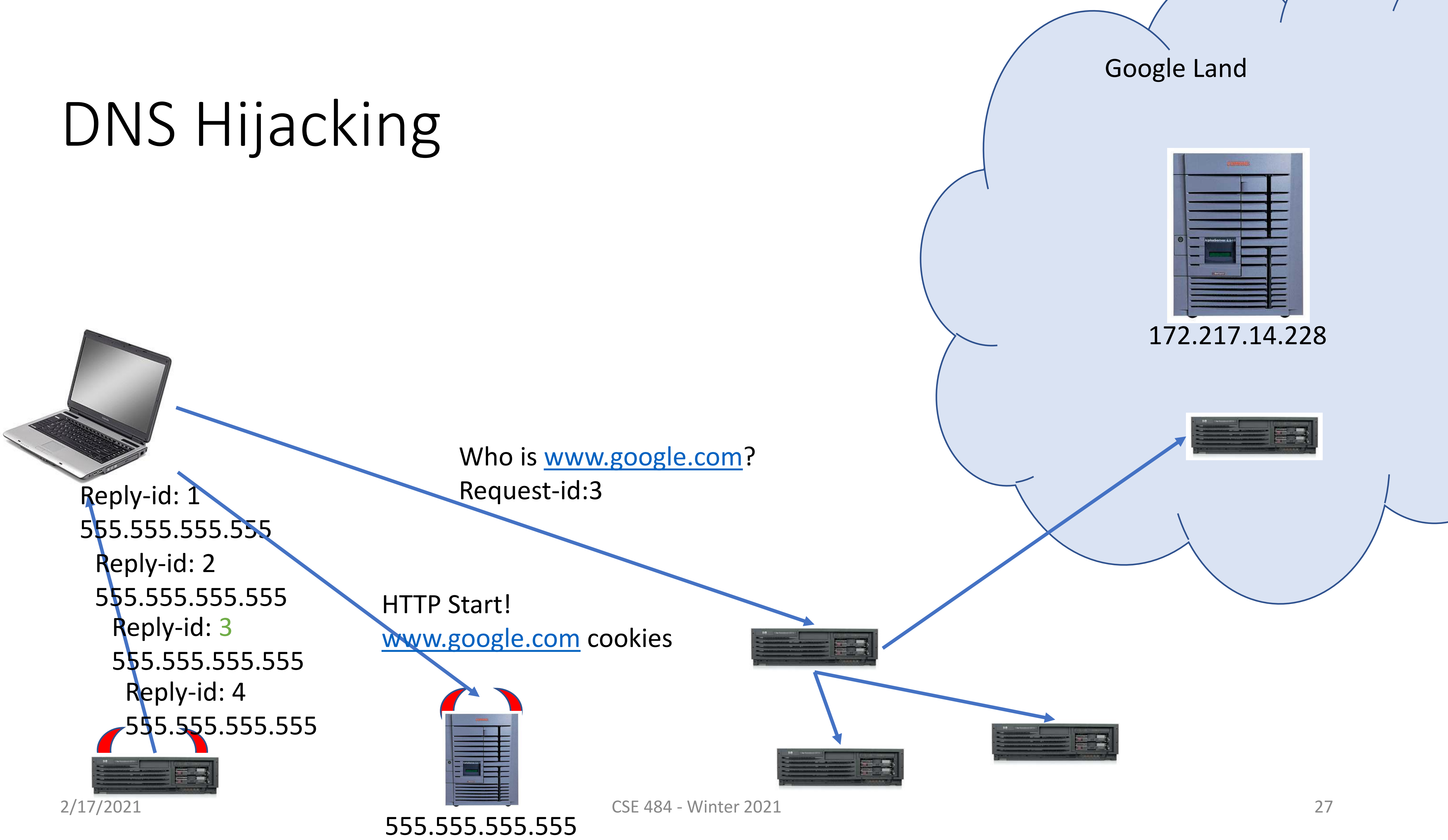


DNS is *unauthenticated* and over UDP

- 16-bit ‘request ID’
  - Used to be *sequential*
  - Now random

- Reply is cleartext and ‘simple’

DNS Hijacking



Throwback: Birthday Paradox

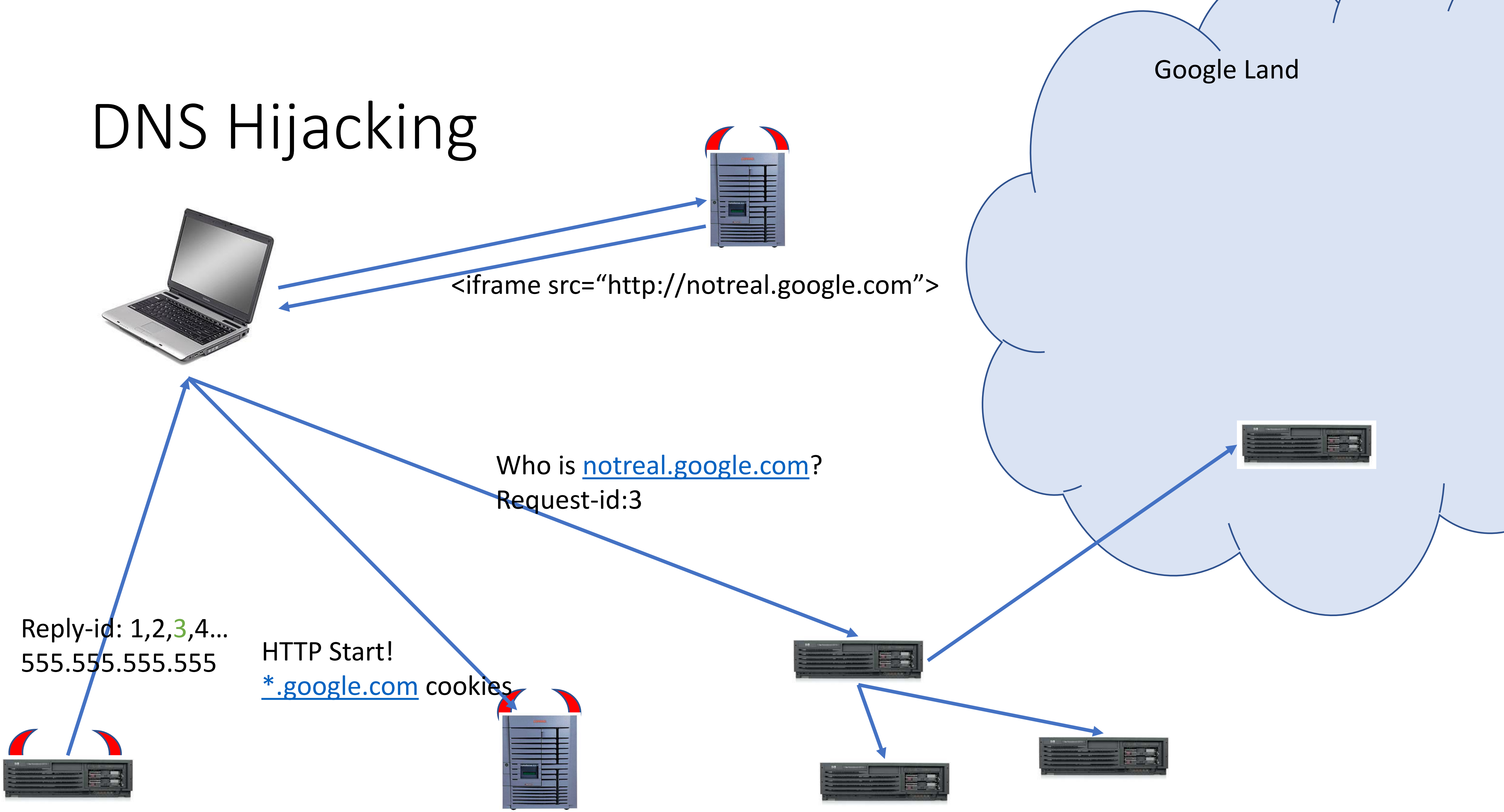
- Are there two people in the first 1/8 of this class that have the same birthday?
  - 365 days in a year (366 some years)
    - Pick one person. To find another person with same birthday would take on the order of 365/2 = 182.5 people
    - **Expect birthday "collision" with a room of only 23 people.**
    - For simplicity, approximate when we expect a collision as **sqrt(365)**.
- Why is this important for cryptography?
  - 2^{128} different 128-bit values
    - Pick one value at random. To exhaustively search for this value requires trying on average 2^{127} values.
    - **Expect "collision" after selecting approximately 2^{64} random values.**
    - **64 bits** of security against collision attacks, not 128 bits.

DNS Hijacking Continued

- 16-bit ID: 2^8 for collision (256!)

- How do we get the victim to as for www.google.com?
  - How about “notreal.google.com” instead?

DNS Hijacking



Google Land

<iframe src=“http://notreal.google.com”>

Who is notreal.google.com?
Request-id:3

Reply-id: 1,2,3,4…
555.555.555.555

HTTP Start!
*.google.com cookies

555.555.555.555

The state of DNS

- Randomize:
  - Request ID
  - **Port number**

- … hope!

Network security

- All our protocols weren’t built for security ☹

- DNS
- BGP
- DHCP
- …