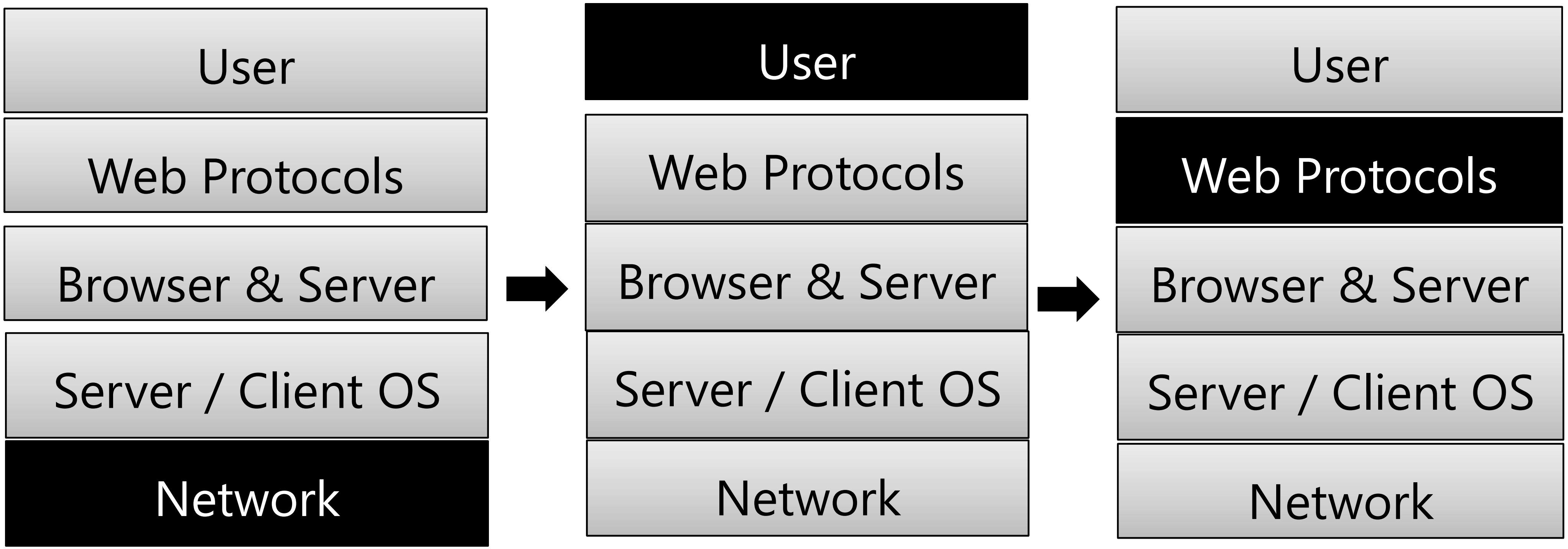
Web Security: User Authentication & Authorization

Prateek Saxena

Recap: Threat Models



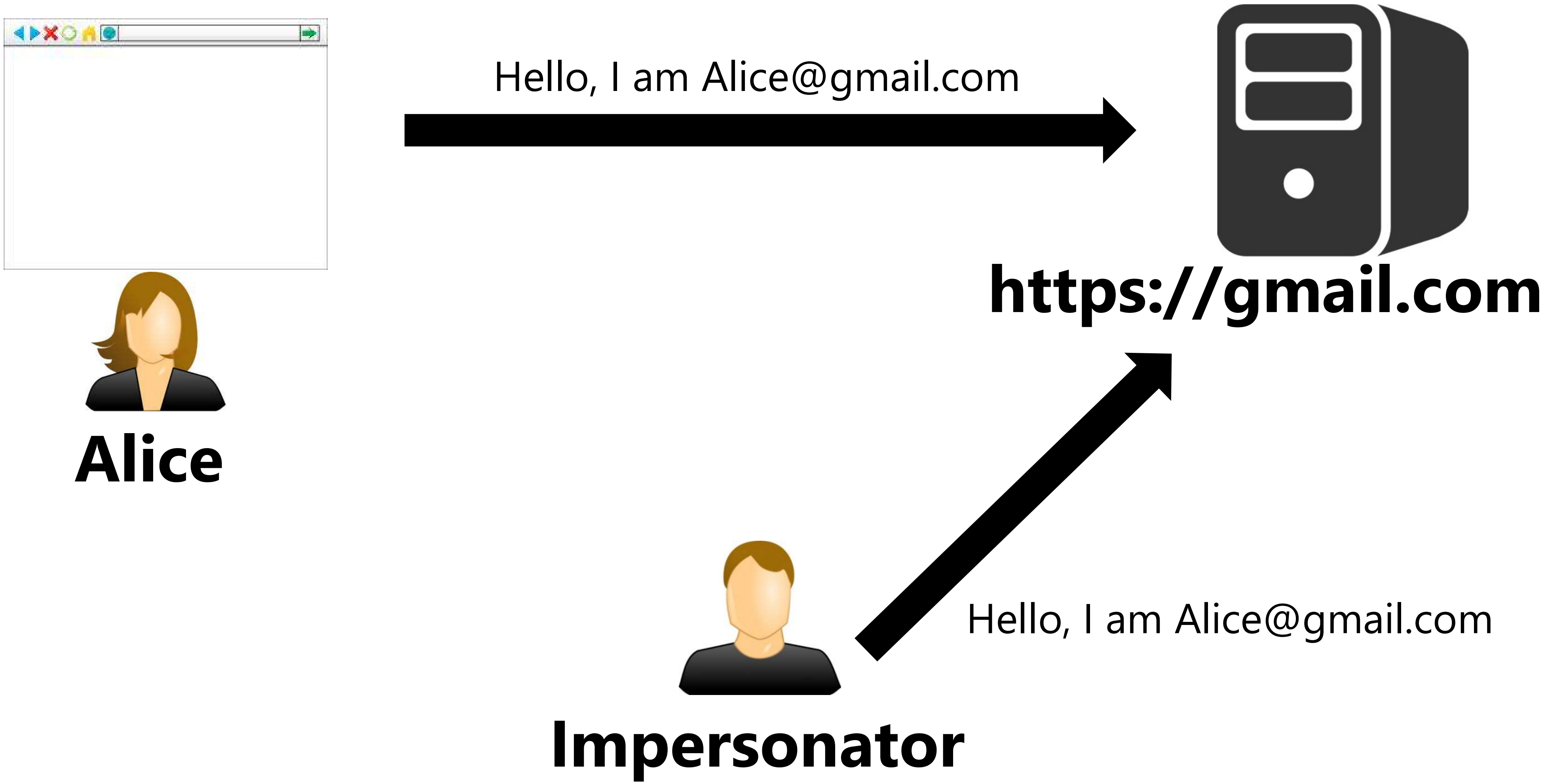
So far... Today...

The Web Attacker Threat Model

- Strictly weaker than a network attacker
- **Web Attacker (Definition)**
  - Owns a valid domain, server with an SSL certificate
  - Can entice a victim to visit his site
    - Say via "Click Here to Get a Free iPad" link
    - Or, via an advertisement (no clicks needed)
  - Can't intercept / read traffic for other sites.
- Assumptions:
  - Network channel is secure
  - Browser is secure

Authentication Protocols

Threat Model: Attacker's Goal



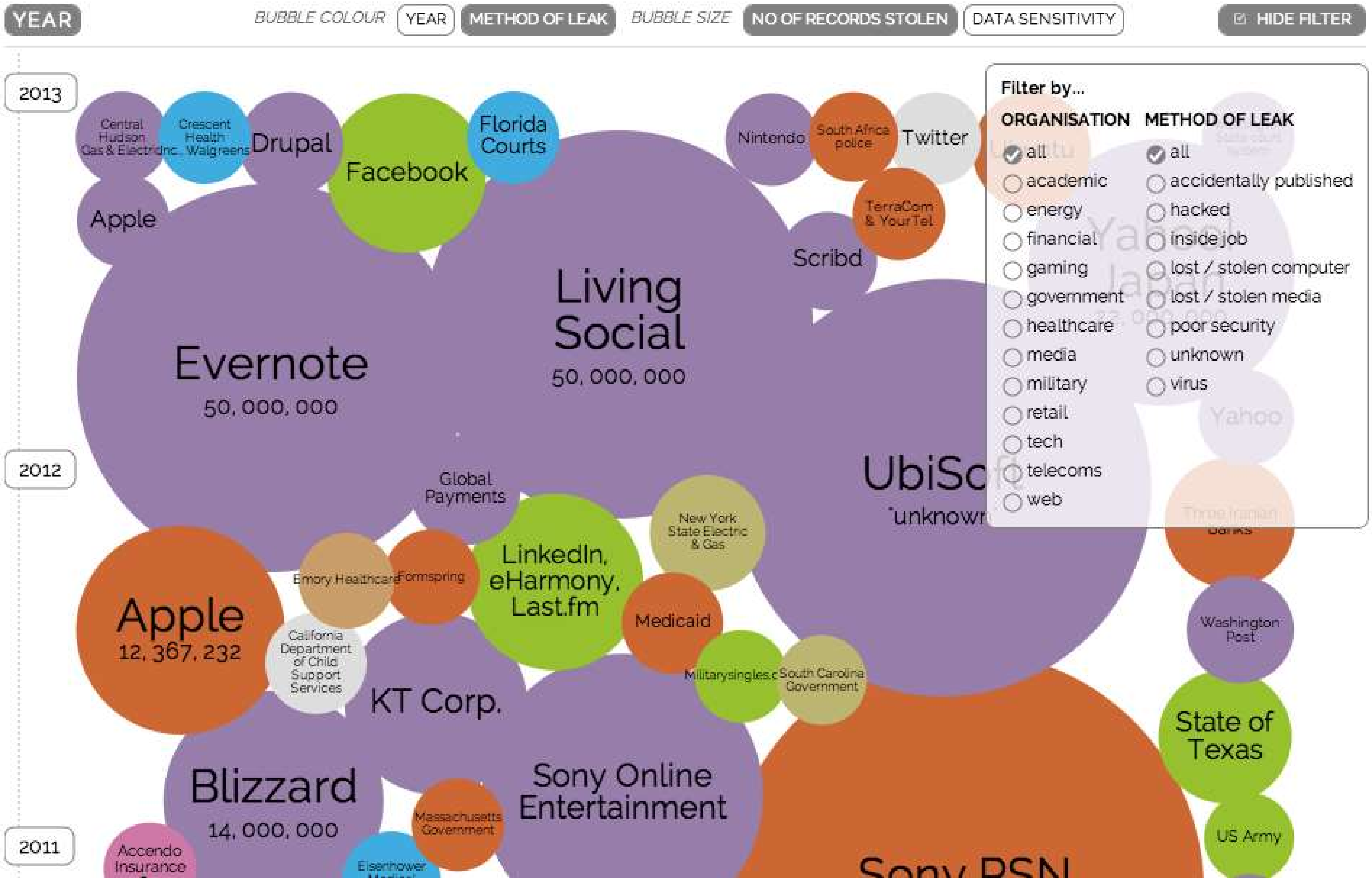
TOFU: 'Trust on First Use'

- The idea:
  - Alice and Gmail exchange a certificate or password the **first** time
  - Gmail **blindly trusts** the user the first time.
  - Subsequent visits are authenticated using the trusted cert. or password

- Used in many systems [see Wikipedia]
  - SSH, GitHub, etc.
  - Self-signed certificates as in SSL's mutual auth (mTLS)

Password-based authentication

Password Data Breaches



[DataBreaches (Information is Beautiful [Website])]()

Server-side Password Hashing

- Passwords shouldn't be stored in plaintext
- Attack 1: A 'smash-and-grab' attack on Gmail gives the imp. Gmail's password database
- Solution 1: GMail stores **H(pwd)**
  - Why hash?
    - Hash functions have a 'one-way' property

- Attack 1: The imp. brute-forces to match Alice's hash
- Attack 2: The imp. Brute-forces to match any hash in Gmail's DB
- Many password cracking tools exist [e.g. JTR]
  - Start with a known set of words, combine them using rules
    - E.g. concatenate, replace "e" with "3", etc.

- Weakness in Solution 1: Attacker can pre-compute a dictionary of (pwd, H(pwd)) and reuse across all sites
- Solution 2: Salt – and – hash
  - **H (r || pwd)** // r can be stored in plaintext
  - Same effort to crack one password
  - Dictionary-based attacks are harder on a list (can't reuse guesses)

Password Recovery

- Common: “Secret” Questions?
  - Name your pet, Aunt’s middle name, Movie…
  - Problem: Not really secret!

- **[Optional]** Reading: Your Pa$$word doesn't matter

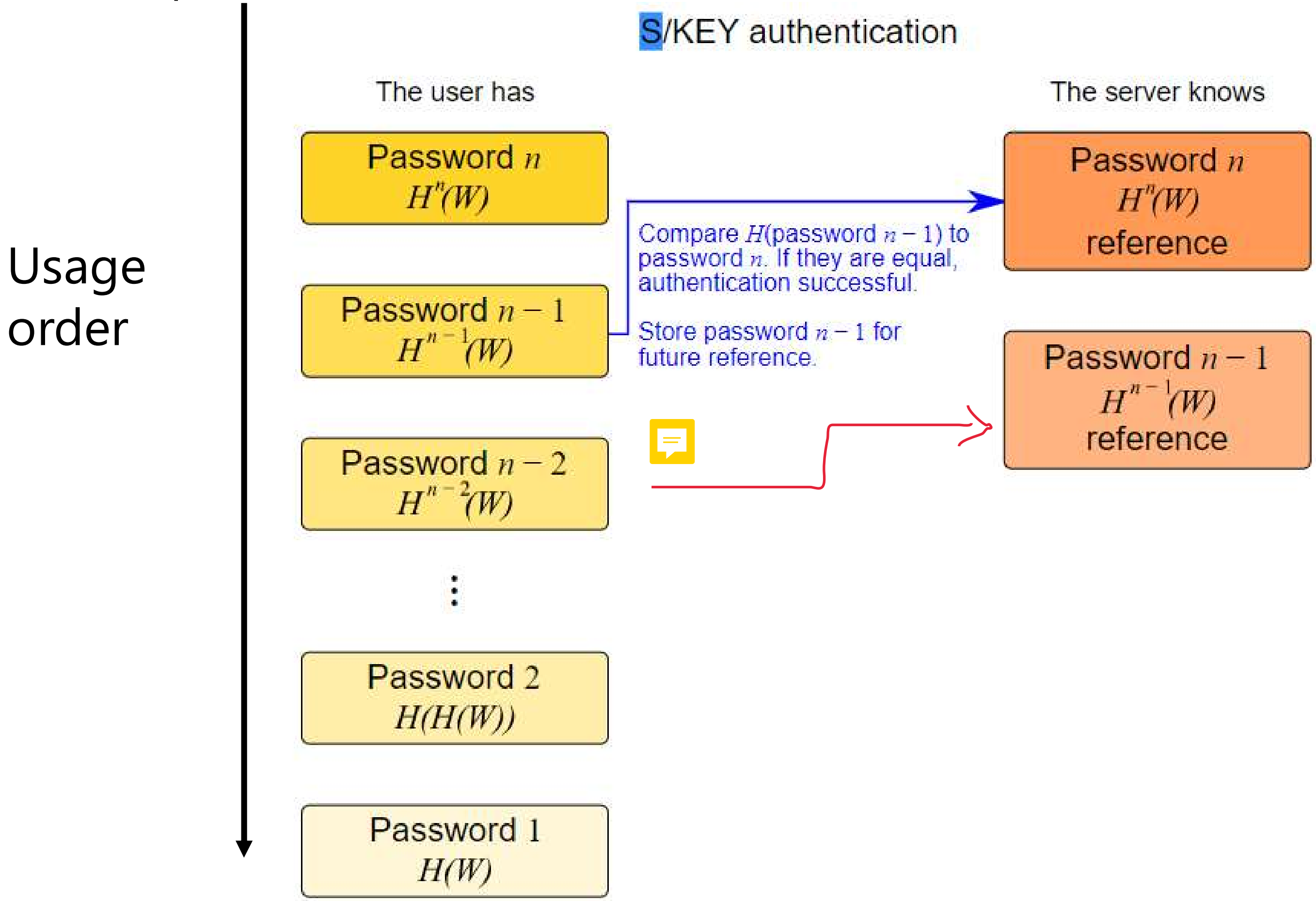
Two-factor Authentication (2FA)

- How to authenticate?
  - Something you know
  - Something you are
  - *Something you have*

- Two-factor Authentication
  - Sending secrets via a second channel (e.g. a special device assumed to be uncompromised)
  - Pros: Added factor of security!
  - Cons:
    - Is it really an additional factor?
    - Easy to use?

Two-factor Authentication (2FA): Lamport's scheme or S/Key

- One possible scheme: S/Key (or Lamport's scheme)
  - Can the adversary guess $H^{n-1}(W)$ from knowing $H^{n}(W)$?
  - Which attacks does this scheme defeat?
    - 'Smash and grab' on server? On user's device? Persistent instead of 'smash and grab'?
    - Interception on network is *not* in the threat model

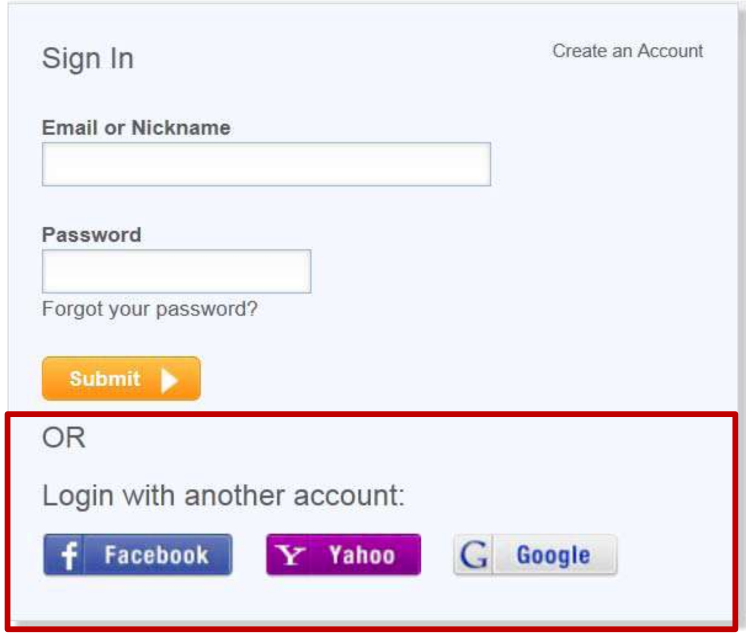


Delegated Authentication

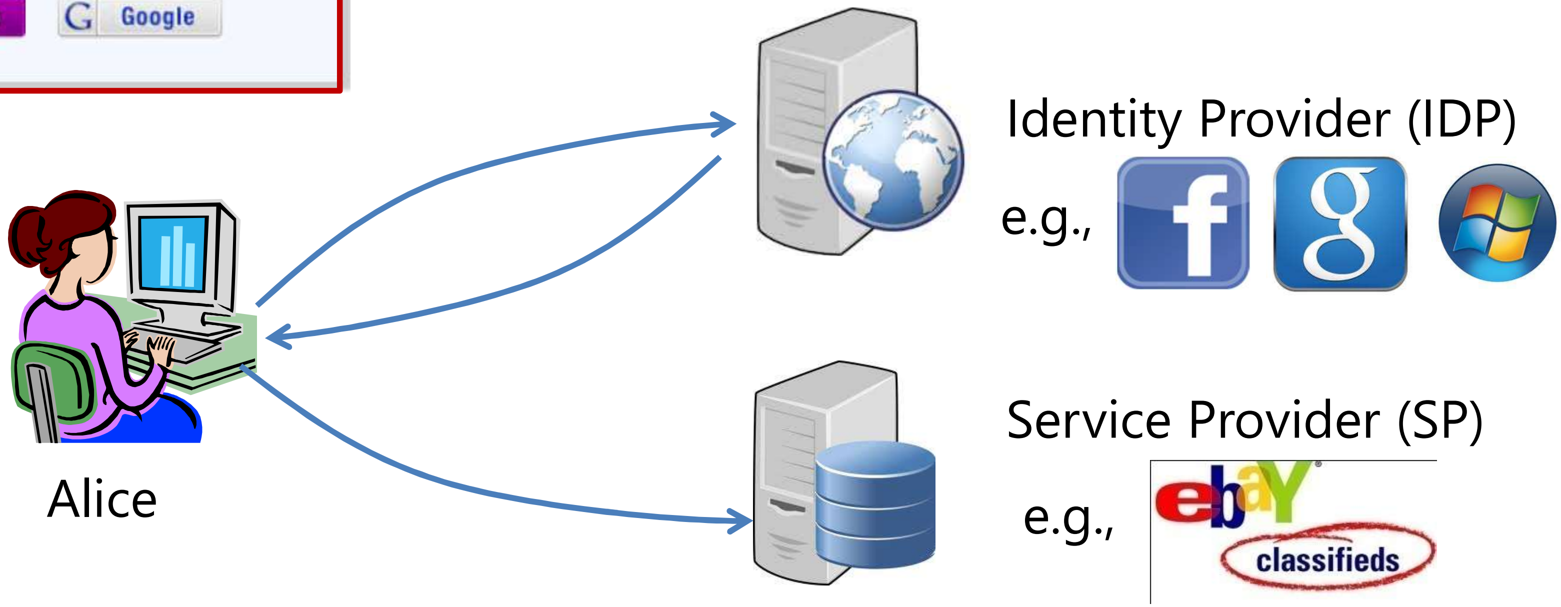
The Idea Of Single Sign-On (SSO)

- Login once, and authenticate everywhere
- Examples
  - Kerberos, OpenID, etc..
  - Usage: Facebook Connect, Google Login, etc…

- Web Authentication

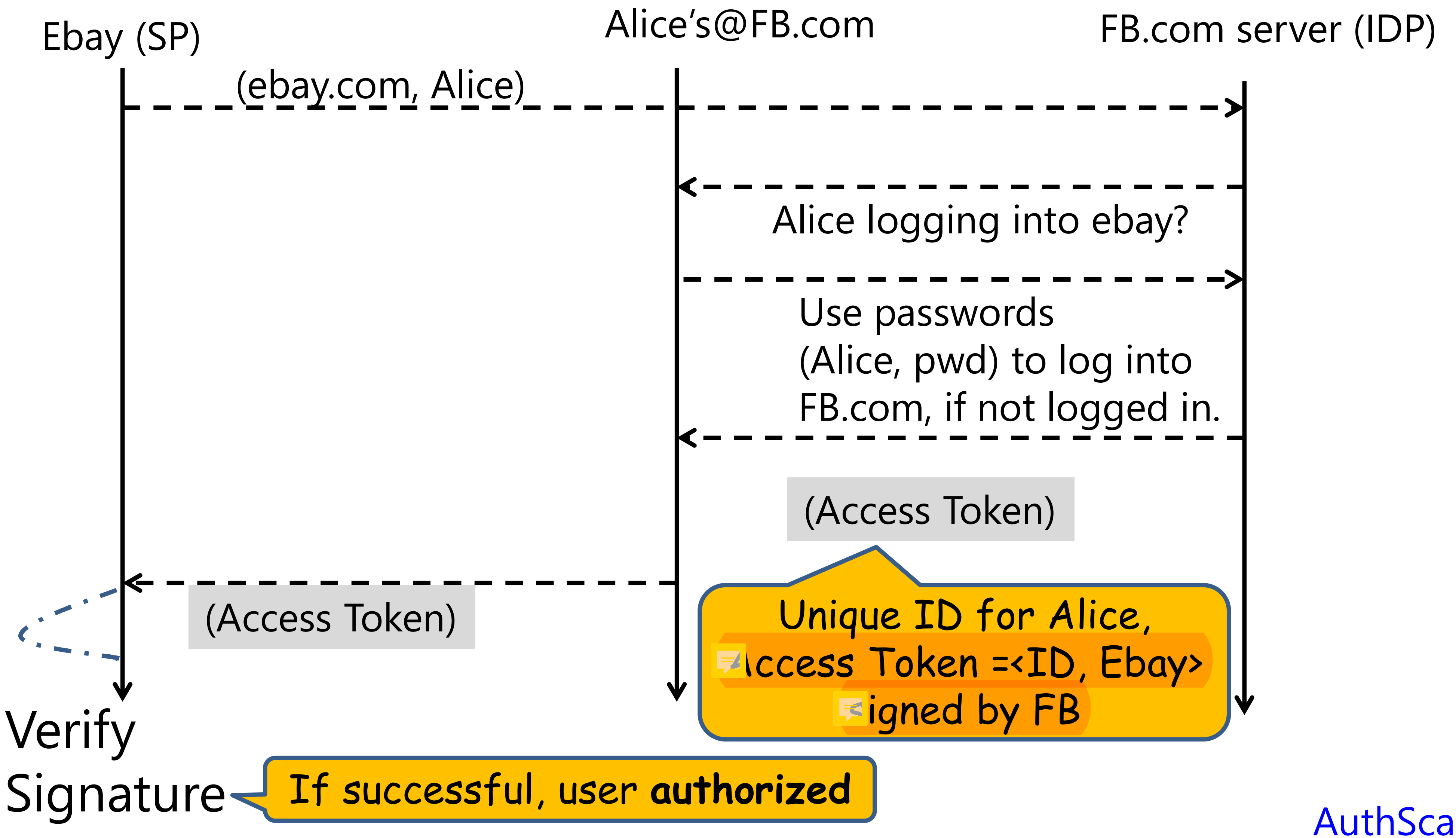


- Single Sign-On (SSO)
  - BrowserID (Mozilla)
  - Facebook Connect
  - Google Login
  - OpenID...



How Does It Work?

- E.g. Simplified Overview of FB Connect



[AuthScan]()

Tightening the threat model

- Consider the web attacker model
- What threats should SSO protocol defeat?
  - Impersonation by IDP / SP
    - FB can become a web attacker and impersonate Alice
    - Ebay can impersonate Alice
  - Limiting the chain of delegation
    - Alice wants authorization to access ebay.com, but silently also get authorization to access evil.com…
- Concerns outside the Threat Model:
  - Linking identities across services?
    - Privacy
  - Bugs in the protocol? Its Implementation in a site?
  - Browser bugs

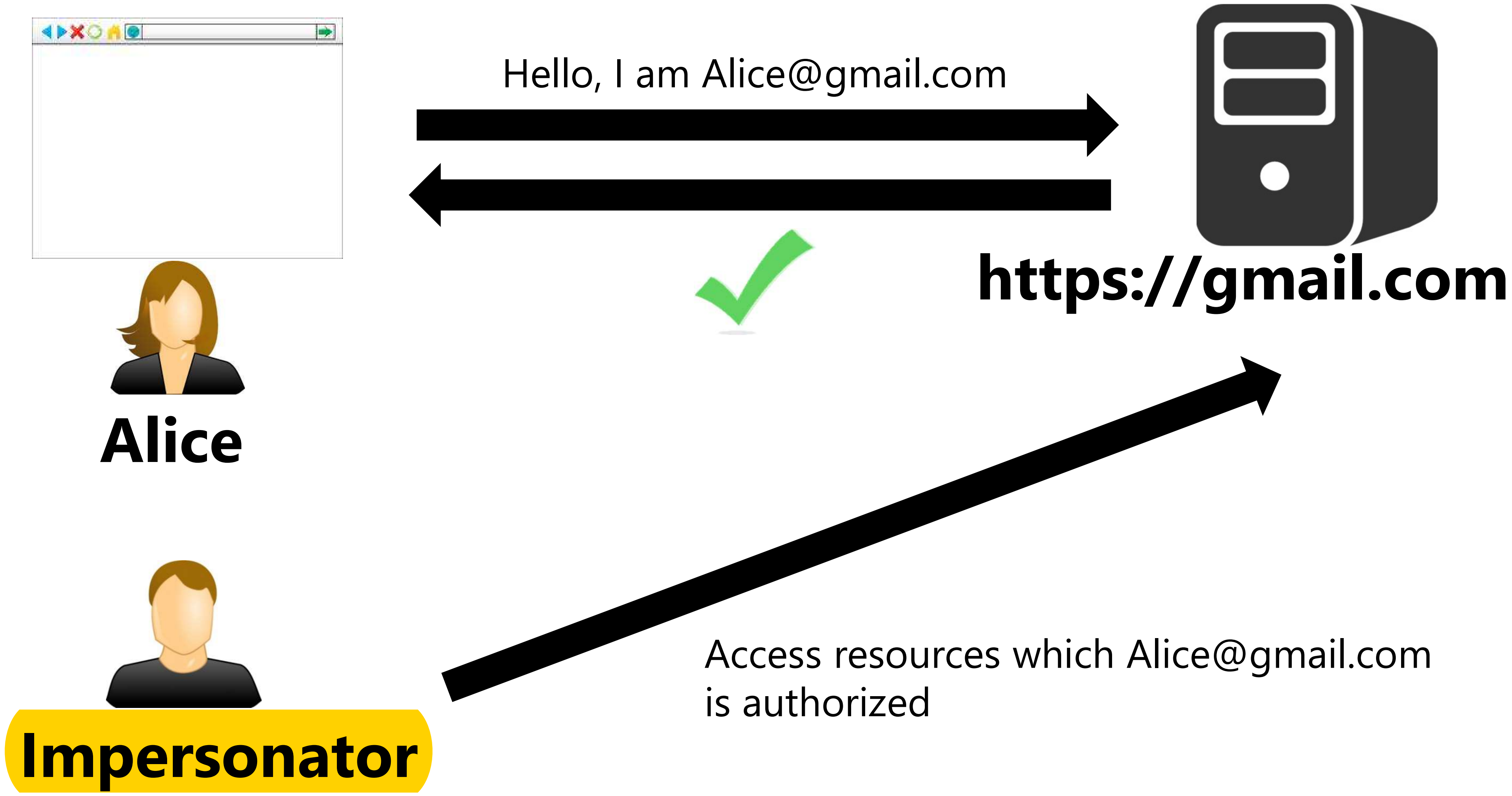
Web Request Authorization

Authentication vs. Authorization

- Authentication
  - To check if A is who they claims to be
  - Protocol ends in a “yes” / “no”, certificate, or token

- Authorization
  - To allow A to access resources hosted at E
  - Protocol uses the output of authentication protocol (e.g. certification, auth. Token, cookie, etc.)

Threat Model - Authorization Protocols



* The attacker (impersonator) is the standard web attacker.

Background:
Web Basics

The Web Platform

Browser

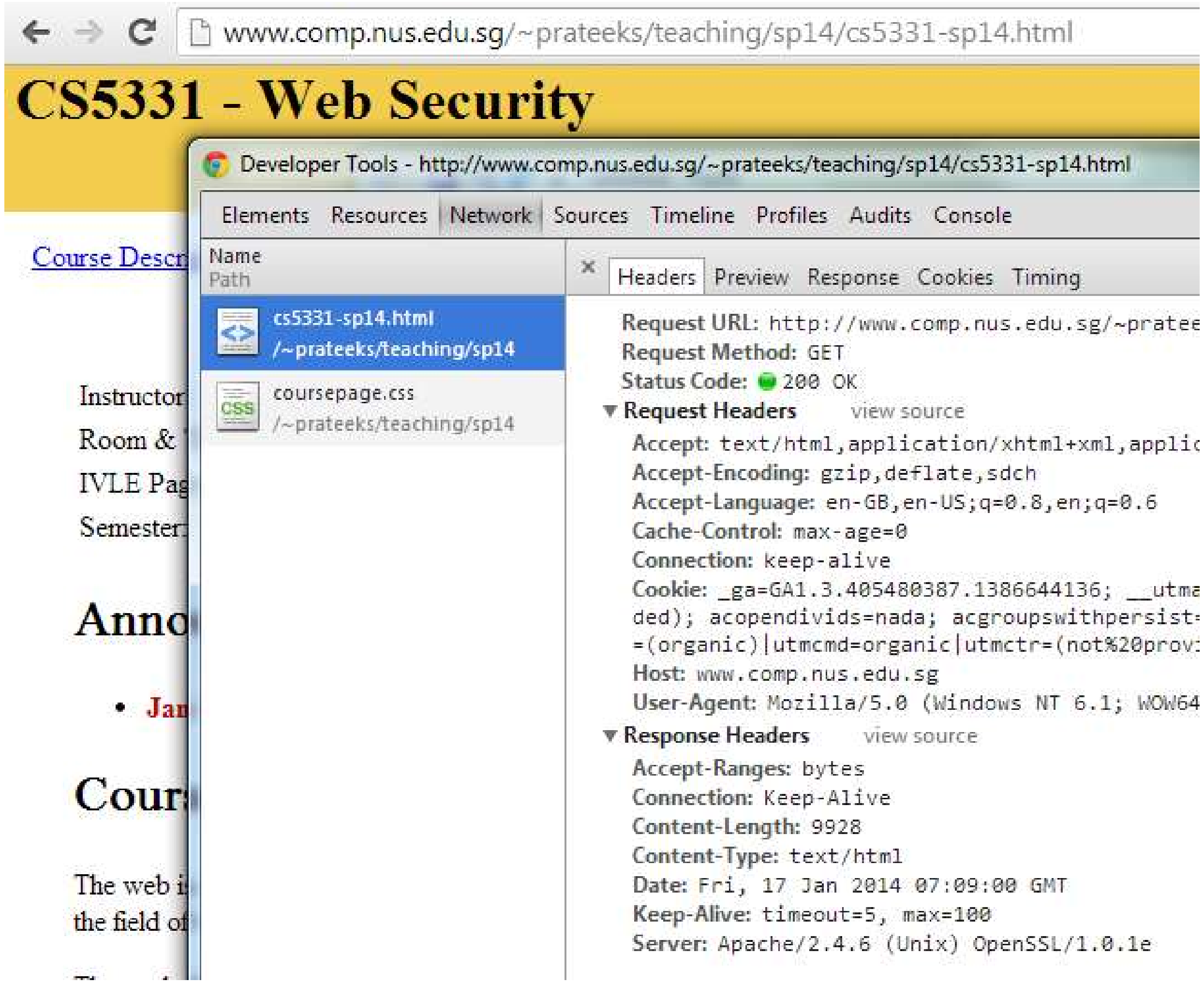
Extensions

Application Protocols

Web Frameworks

HTTP


Bank


www.comp.nus.edu.sg/~prateeks/teaching/sp14/cs5331-sp14.html
CS5331 - Web Security
Developer Tools - http://www.comp.nus.edu.sg/~prateeks/teaching/sp14/cs5331-sp14.html
Elements Resources Network Sources Timeline Profiles Audits Console
Name
Path
cs5331-sp14.html
/~prateeks/teaching/sp14
coursepage.css
/~prateeks/teaching/sp14
Headers Preview Response Cookies Timing
Request URL: http://www.comp.nus.edu.sg/~pratee
Request Method: GET
Status Code: 200 OK
Request Headers view source
Accept: text/html,application/xhtml+xml,applic
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-GB,en-US;q=0.8,en;q=0.6
Cache-Control: max-age=0
Connection: keep-alive
Cookie: _ga=GA1.3.405480387.1386644136; __utma
ded); acopendivids=nada; acgroupswithpersist
=(organic)|utmcmd=organic|utmctr=(not%20prov
Host: www.comp.nus.edu.sg
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64
Response Headers view source
Accept-Ranges: bytes
Connection: Keep-Alive
Content-Length: 9928
Content-Type: text/html
Date: Fri, 17 Jan 2014 07:09:00 GMT
Keep-Alive: timeout=5, max=100
Server: Apache/2.4.6 (Unix) OpenSSL/1.0.1e
Course Descr
Instructor
Room &
IVLE Pag
Semester
Anno
Jan
Cour
The web i
the field of

Frames / Windows

- Each window is a frame
  - A frame hosts a web origin (see SOP)
- Iframes: Inline frame
  - Can host a different origin

```
<!DOCTYPE html>
<html>
<body>

<iframe src="http://www.w3schools.com">
  <p>Your browser does not support iframes.</p>
</iframe>

<script>
document.write (frames[0].parent.location.href);
</script>

</body>
</html>
```

Frame Navigation

- Can be “navigated” by
  - User typing in the URL bar, user clicks links
  - Using scripts

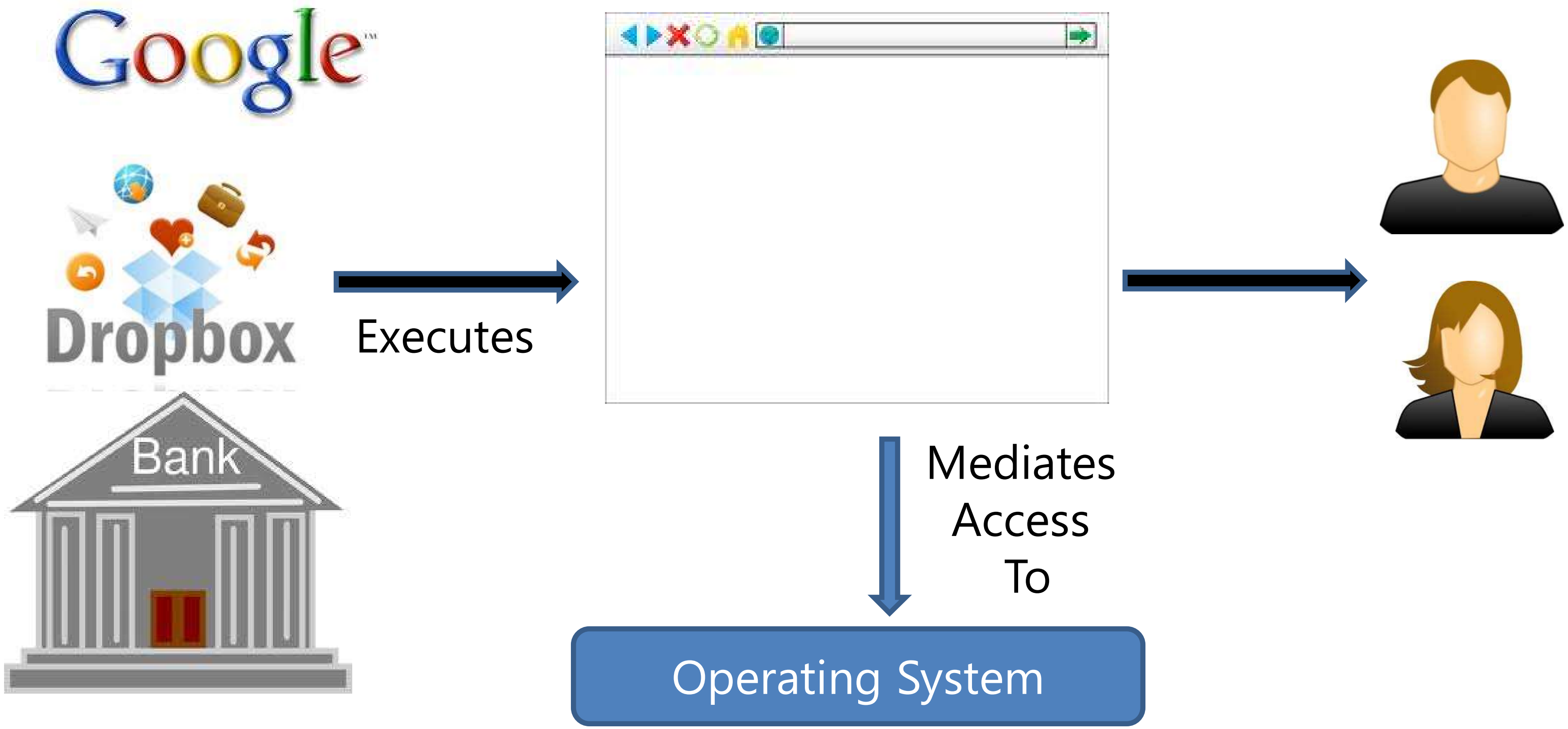
```
<iframe src="http://www.w3schools.com">
  <p>Your browser does not support iframes.</p>
</iframe>

<script>
frames[0].location = "http://www.comp.nus.edu.sg/~prateeks/teaching/sp14/cs5331-sp14.html";
</script>
```

Recall The Web Attacker Threat Model

- Strictly weaker than a network attacker
- **Web Attacker (Definition)**
  - Owns a valid domain, server with an SSL certificate
  - Can entice a victim to visit his site
    - Say via "Click Here to Get a Free iPad" link
    - Or, via an advertisement (no clicks needed)
  - Can't intercept / read traffic for other sites.
- Assumptions:
  - Network channel is secure
  - Browser is secure

Security Goals of a Web Browser



Web Apps **Resources** **Users**

Security Goals of a Web Browser

- 2 Kinds of Isolation
  - Prevent network content to access OS resources
    - E.g. Installing EXEs, Camera, GPS,…
  - *Isolate Web Sites from each other*
    - *Via the "same origin policy"*

The Same Origin Policy

No direct access between browser frames !

https://**www.nus.edu.sg**:**443**/~prateeks/add.php?q=x

PROTOCOL

HOST (Domain)

PORT

WEB ORIGIN = PROTOCOL + HOST+ PORT

1. [Same-origin policy [Wikipedia]]
2. [RFC 6454]

The Technical Detail in SOP

- **P0:** Sub-domains can access parent domain
  - Unless parent is a "public suffix" (e.g. .edu, .com, ...)
- **P1:** Can make X-domain HTTP requests for some sub-resources
  - HTTP GET resources like JS, CSS, Images (Cookies are sent!)
  - HTTP POST request to a different domain
  - Cross-frame communication channels... (XHR, CORS, postMessage)
- **P2:** Note on Granularity: Origins (paths excluded)
  - *Is this a good security model?*

http://www.comp.nus.edu.sg/~prateeks/

⇩ *Can run code on*

http://www.comp.nus.edu.sg/

  - *Why is this potentially problematic?*
  - *If you visit my site logged in on nus.edu.sg, I can run code on your behalf and view your scores...*
  - *But, this is how the web standard define it. So, we work with it.*

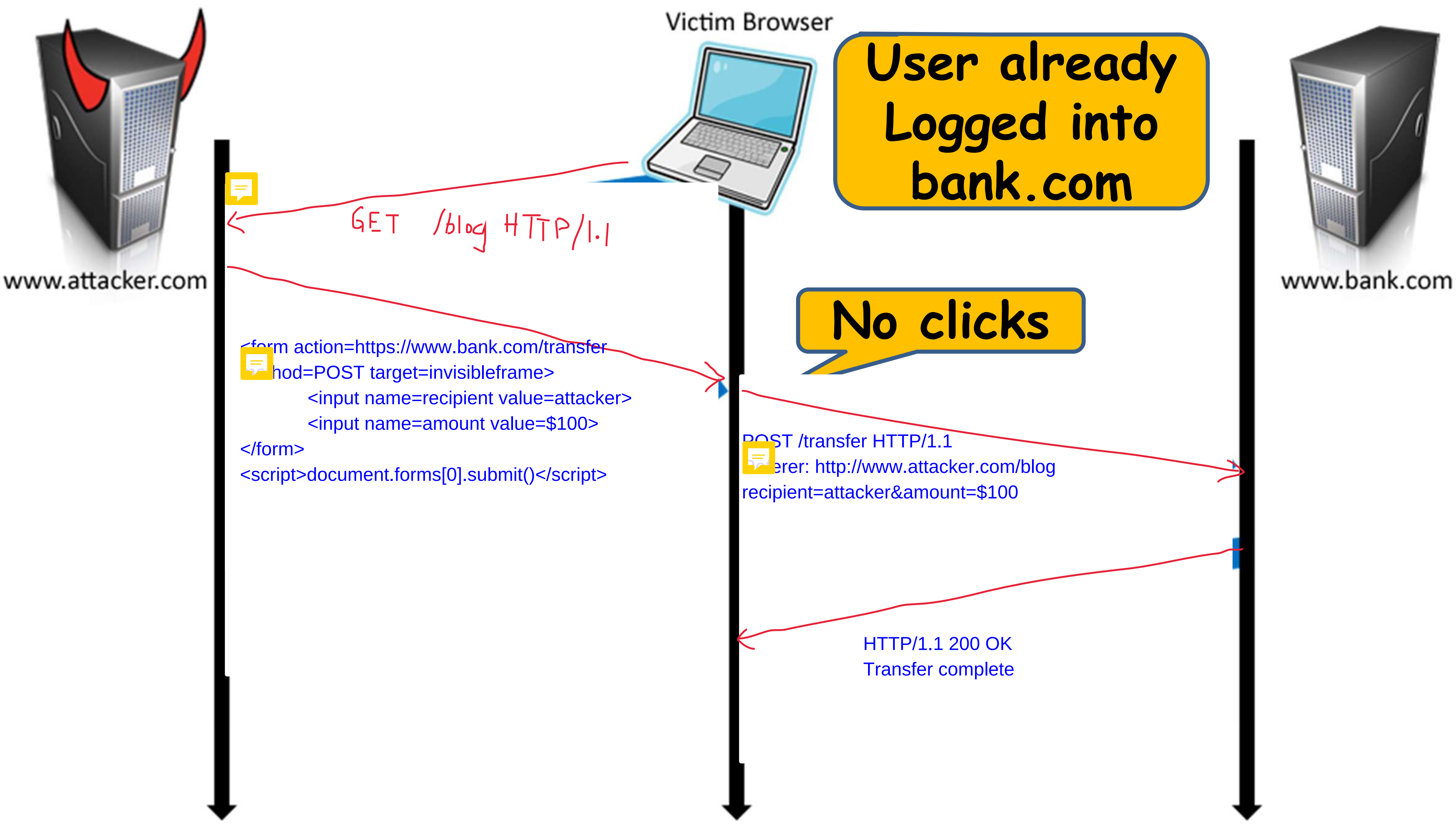
Attacks on Web Authorization: Cross-site Request Forgery

The SOP allows cross-origin HTTP POST requests


```
<form method="post" action="http://bank.com/trasfer">
    <input type="hidden" name="to" value="ciro">
    <input type="hidden" name="ammount" value="100000000">
    <input type="submit" value="CLICK TO CLAIM YOUR PRIZE!!!">
</form>
```

1. The SOP does not prevent origin A from sending HTTP POST requests to B (≠ A)
2. The JavaScript running within the origin A's authority, can auto-submit forms

CSRF Attach: The Main Idea



[CSRF: Dangers, Detection and Defense – C Jackson et. al.]

CSRF Consequences

- Example


EBAY VULNERABLE TO ACCOUNT HIJACKING VIA XSRF

by Michael Mimoso Follow @mike_mimoso December 27, 2013 , 12:02 pm

EBay is vulnerable to a hack that would allow an attacker to hijack an account and make unauthorized purchases from the victim's account that would be difficult to disprove.


CSRF VULNERABILITY PATCHED IN GODADDY DOMAIN SETTINGS

by Michael Mimoso Follow @mike_mimoso January 20, 2015 , 9:50 am

Domain registrar GoDaddy yesterday patched a cross-site request forgery vulnerability that could have allowed an attacker to change domain settings on a site registered with GoDaddy.

Source: ThreatPost1, ThreatPost2

CSRF Defenses

- Secret Validation Token

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

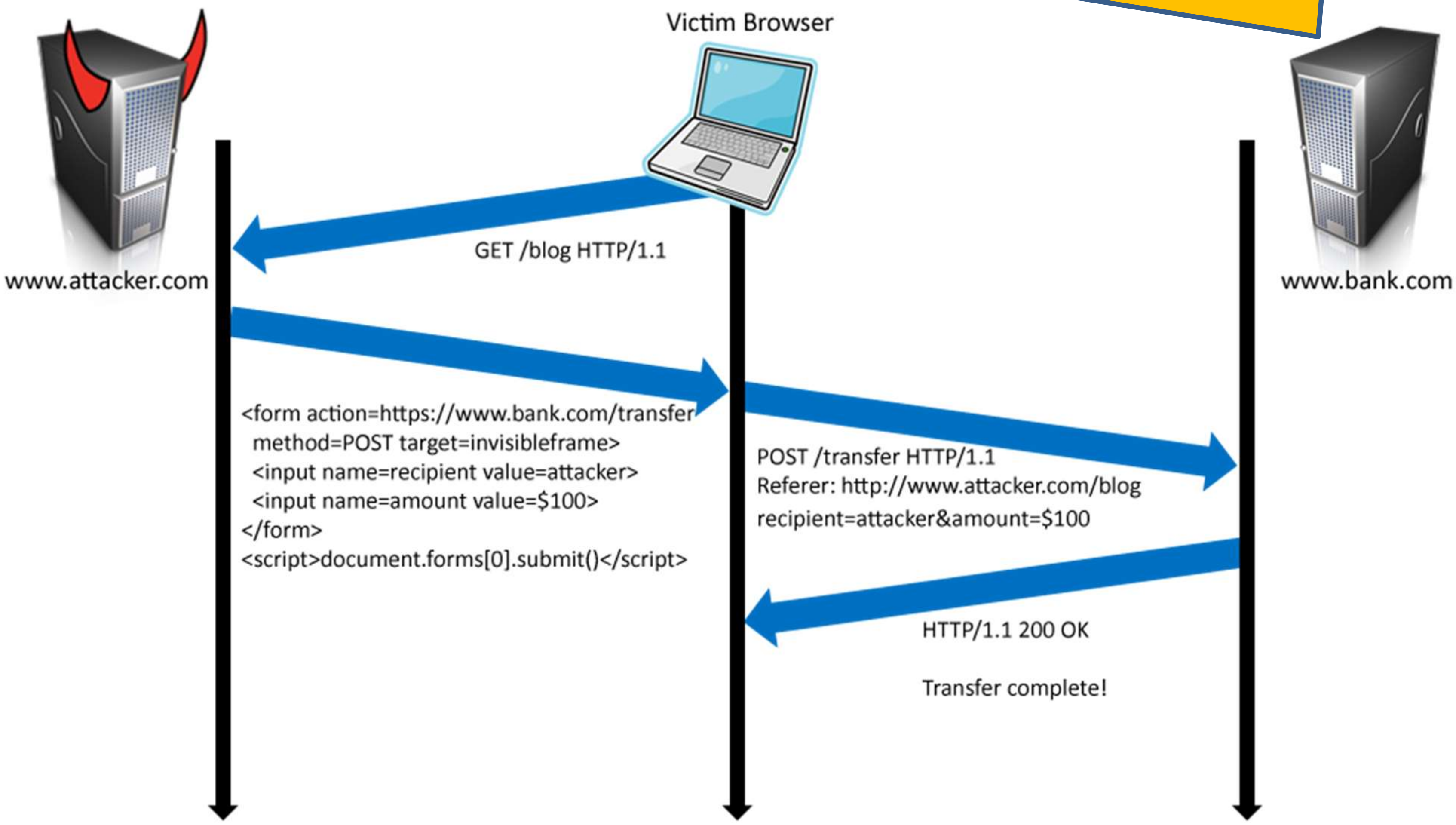
- Referer Validation

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

- Implemented in most common web prog. frameworks

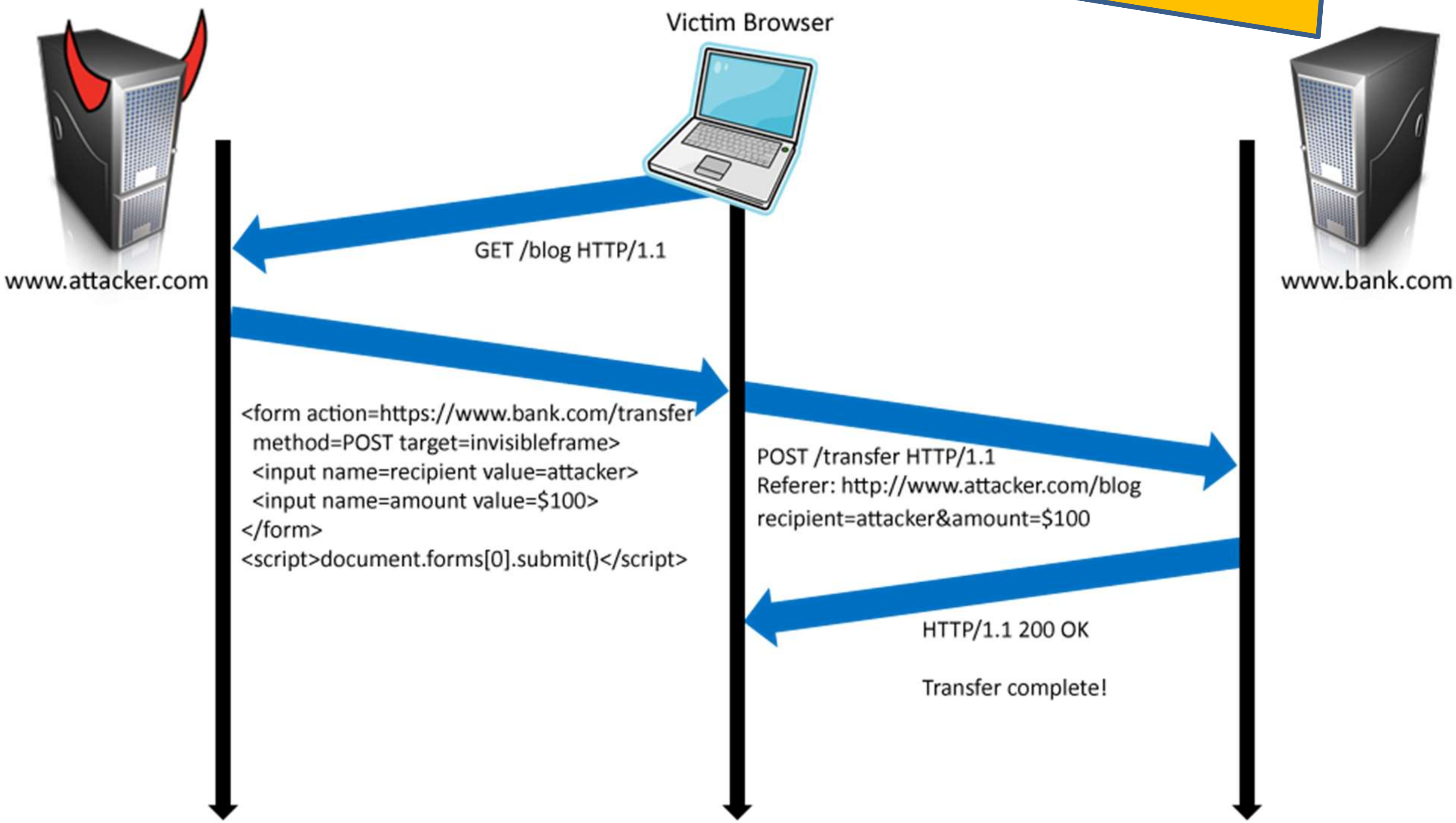
CSRF Defenses

Bank confused the session from which request was made..



CSRF Defenses:
Secret Validation Tokens

Idea #1: Tie the HTTP request to the session…



CSRF Defenses: Secret Validation Tokens

Idea #1: Tie the HTTP request to the session…

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

Random value for each form (Sent in the HTTP POST request)

```
<input type=hidden value=38385abc4>
```

CSRF Defenses:
Secret Validation Tokens

Idea #1: Tie the HTTP request to the session…

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

```
<input type=hidden value=38385abc4>
```

Why does the attack fail?

The Bank server sees the value and decides which session the HTTP request is associated to….

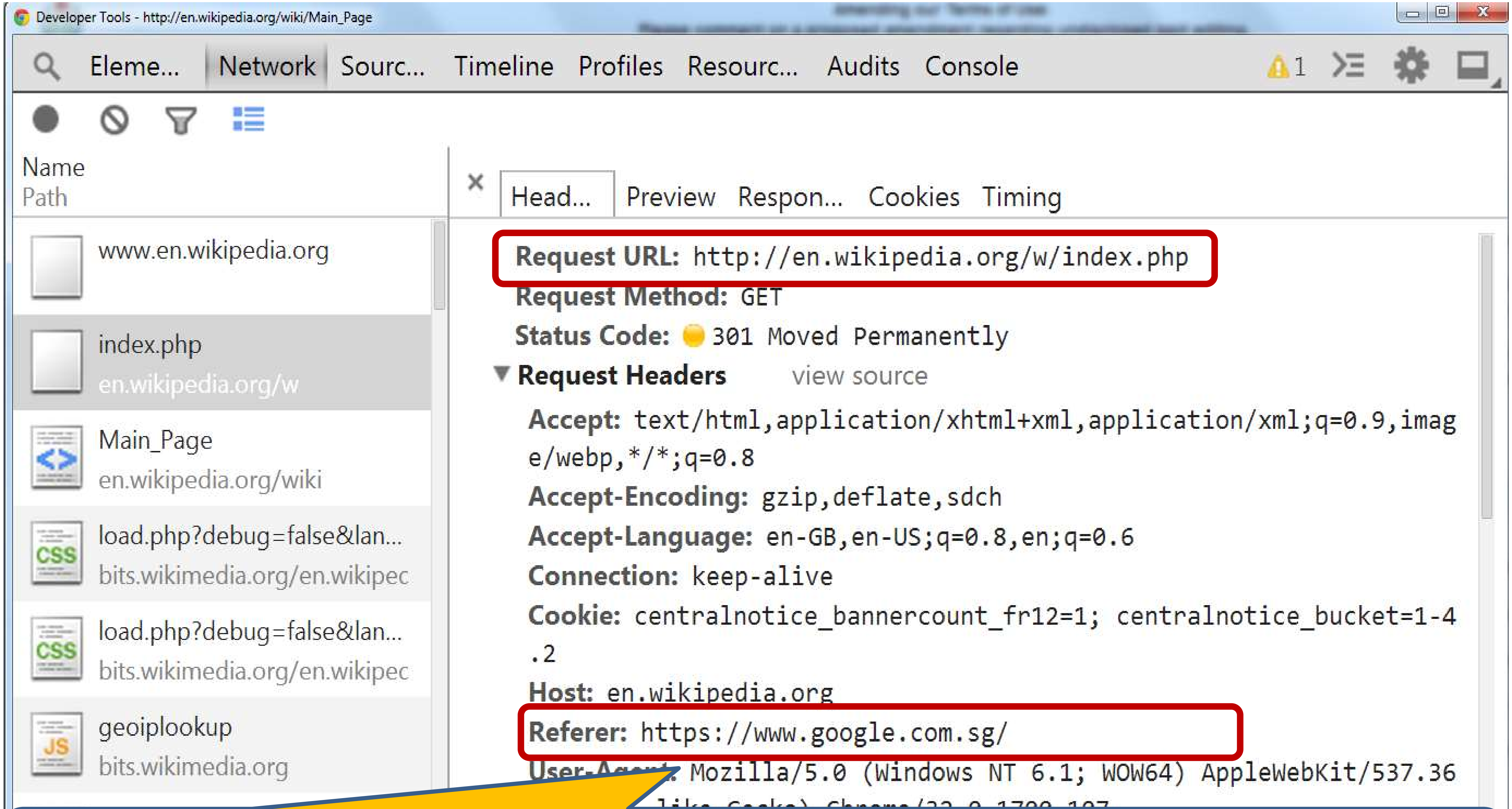
(So, it won’t transfer money on Joe’s behalf)

CSRF Defenses: Secret Validation Tokens

Quiz: What's the best way to generate secret tokens?

- **Hash of User ID**
  - Attacker can forge
- **Session ID**
  - Save-to-HTML does allow session hijacking
- **Session-Independent Nonce (Trac)**
  - Can be overwritten by subdomains, network attackers
- **Session-Dependent Nonce (CSRFx, CSRFGuard)**
  - Requires managing a state table
- **HMAC of Session ID**
  - HMAC (secret_key, sessionID)
  - Best, No extra state required

CSRF Defenses (II): HTTP Referrer Validation



Idea #2: HTTP Referrer tells you which site the request was made from

CSRF Defenses (II): HTTP Referrer Validation

- **Server can check the Referrer to be valid**
  - Attack request comes from http://evil.com
  - Practical caveats:
    - Referrer Headers are stripped off by web sites, network proxies, etc.
      - (HTTPS: < 0.01%, HTTP: 1-3%)
      - So, they don't work in some cases…

Robust Defenses for Cross-Site Request Forgery