Stony Brook University

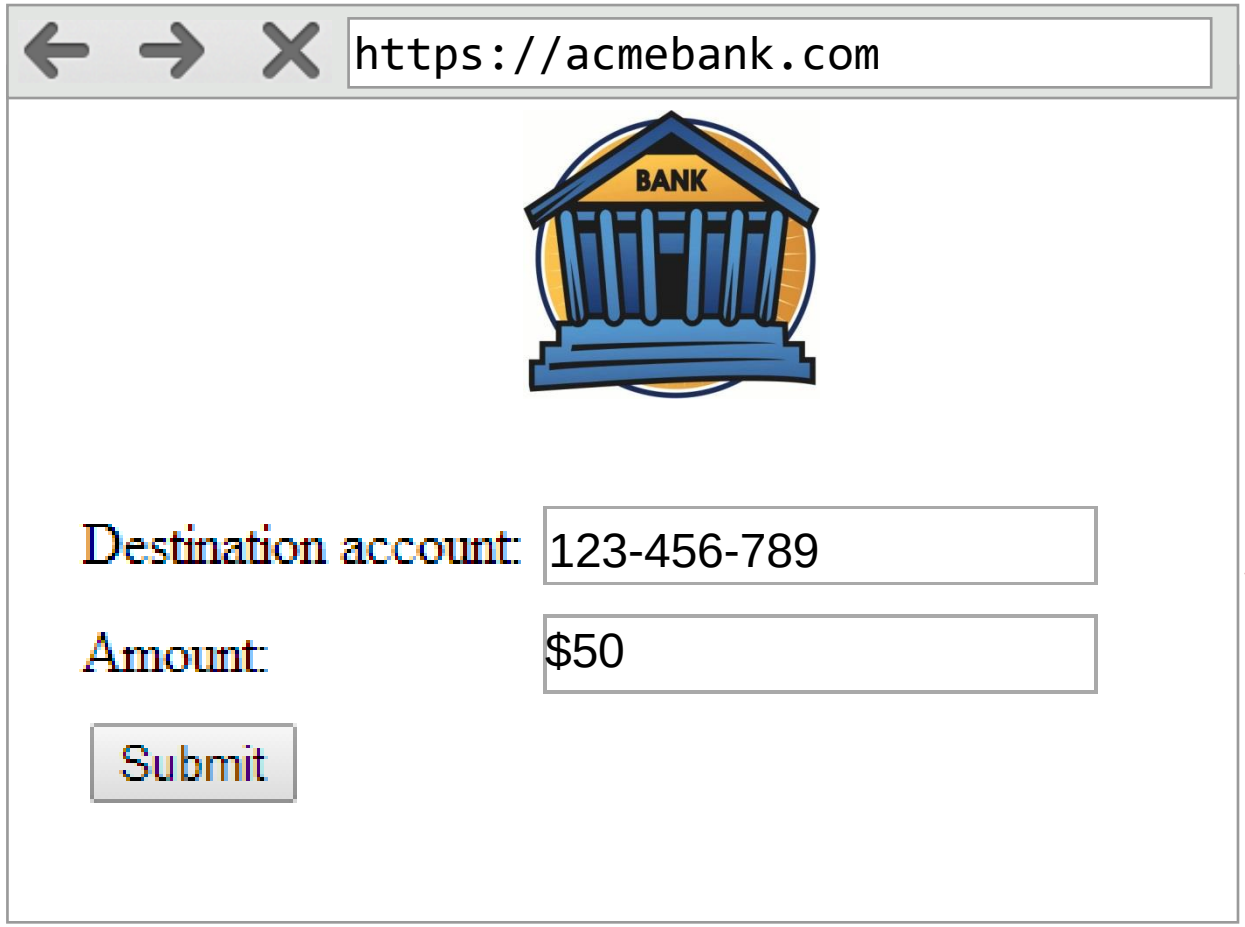
CSE 361: Web Security

CSRF, XSSI, SRI, and Sandboxing

Nick Nikiforakis

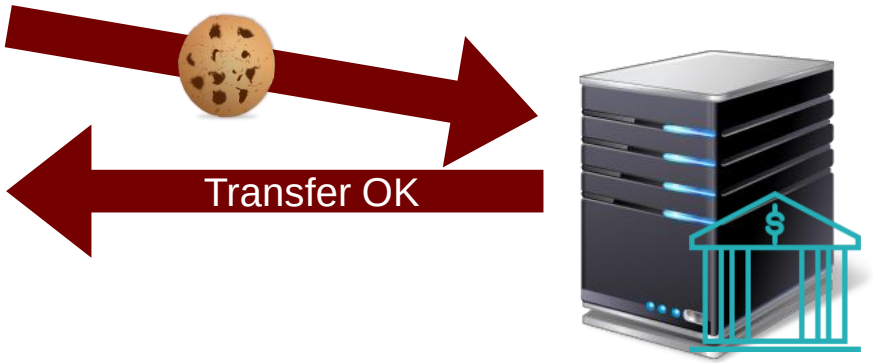
CSRF (Sea Surf)

Regular Web site usage

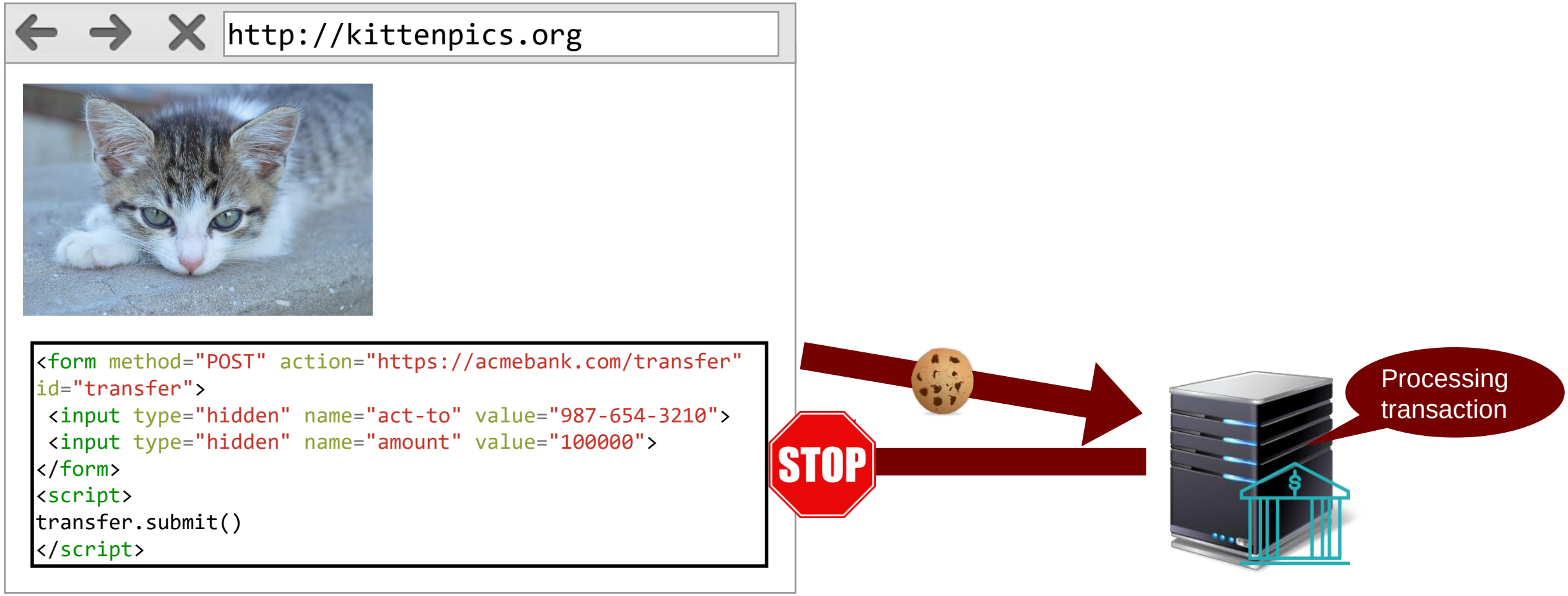


Behind the scenes

```
<form method="POST"
target=https://acmebank.com/transfer>
        <input type="text" name="acct-to">
        <input type="text" name="amount">
        <input type="submit">
</form>
```



Forcing browser to perform an action for the attacker



Cross-Site Request Forgery (CSRF / "Sea Surf")

- Web application does not ensure that state-changing request came from "within" the application itself
- Attack works for GET ...
  - Image tag with src attribute:

```
<img src="https://acmebank.com/transfer?to=attacker&amount=10000">
```

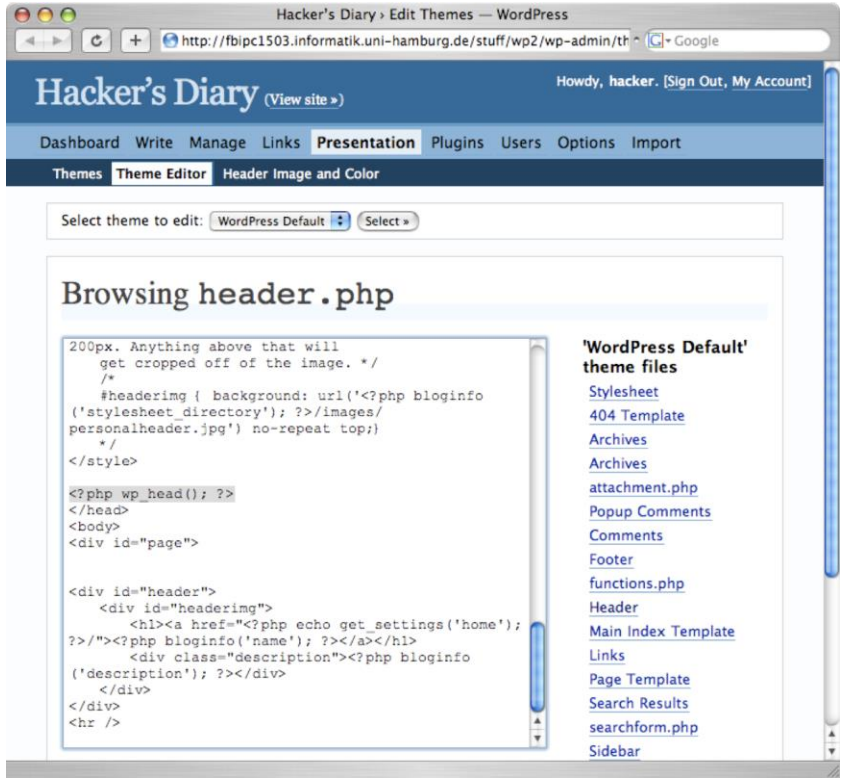
  - Hidden iframes, css files, scripts, ...
- and POST
  - create iframe (or pop-up window)
  - fill created viewport with prefilled form
  - submit form

CSRF Examples: digg.com (2006)

- digg.com determines frontpage based on how many "diggs" a story gets
- vulnerable against CSRF, could be used to digg an URL of the attacker's choosing
- Guess which article made it to the front page...

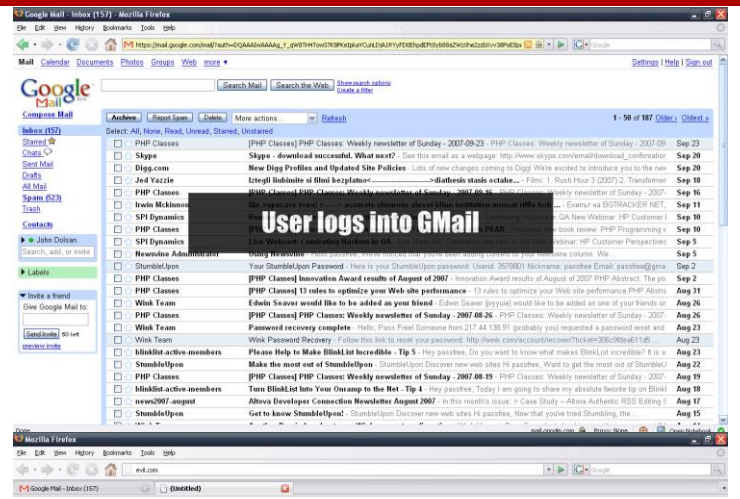
CSRF Example: WordPress < 2.06 (2007)

- WordPress theme editor was susceptible
- WordPress themes are PHP files
- Attacker could modify files when logged-in admin visited his page
  - arbitrary code execution on targeted page

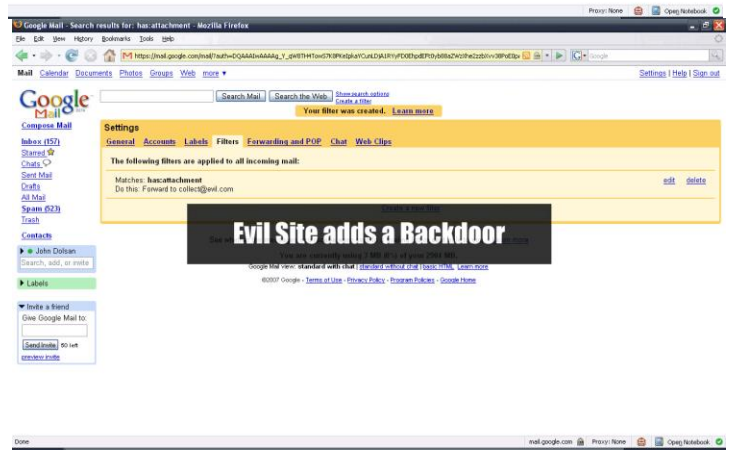


CSRF Example: Gmail filters (2007)

- Google Mail insufficiently protected against CSRF
- Attacker could add mail filters
  - e.g., forward all emails to a certain address
- According to a victim, this led to a domain takeover
  - Attacker adds redirect filter
  - Attacker request AUTH code for domain transfer



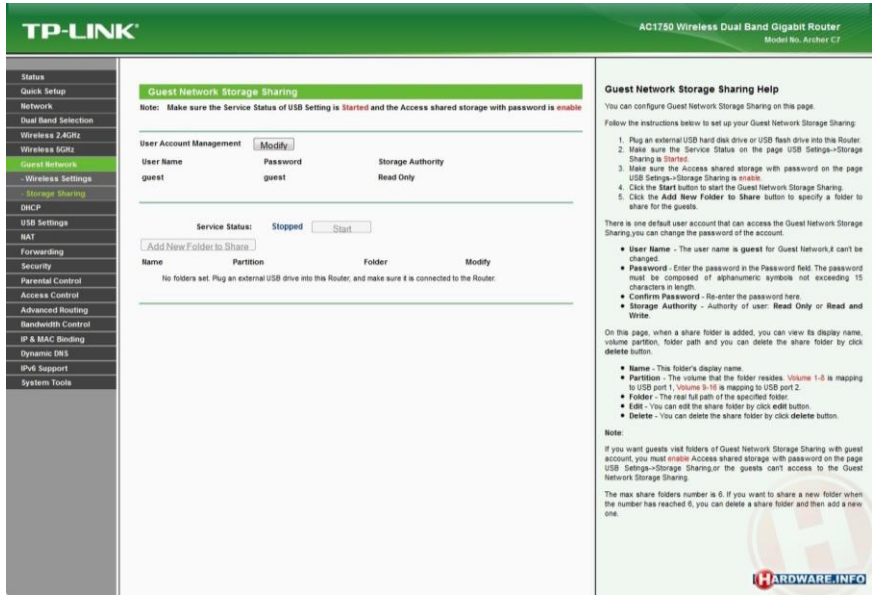
User visits Evil Site



http://www.davidairey.com/google-gmail-security-hijack/

CSRF Example: TP-Link routers (CVE-2013-2645)

- TP-Link Web interface was vulnerable to configuration changes via CSRF
  - set root of built-in FTP server, enable FTP via WAN, ...
  - modify DNS server
- Exploited in the wild to change DNS server
  - redirects all DNS traffic to attacker's server
    - leaking all visited domains
    - allowing for trivial MitM attacks
- Only worked when user was logged in



CSRF in 2017 to 2019

- CVE-2017-7404 D-Link router, firmware upload possible
- CVE-2017-9934 Joomla! CSRF to XSS
- CVE-2018-100053 LimeSurvey Delete Themes
- CVE-2018-6288 Kaspersky Secure Mail Gateway Admin Account Takeover
- CVE-2019-10673 WordPress CSRF to change admin email, password recovery for full compromise

(Not really) Preventing CSRF: Refer(r)er Checking

- CSRF entails cross-domain requests
  - in theory, these should carry a referrer
  - server could decide based on header
- In practice, there are several problems
  - Middleboxes/proxies might strip (complete URL is sent, privacy concerns)
  - Attacker may strip Referer header by
    - using a `data:` URL
    - Referrer-Policy header
- Utility vs. Security trade-off
  - what do we do when the header is not present?

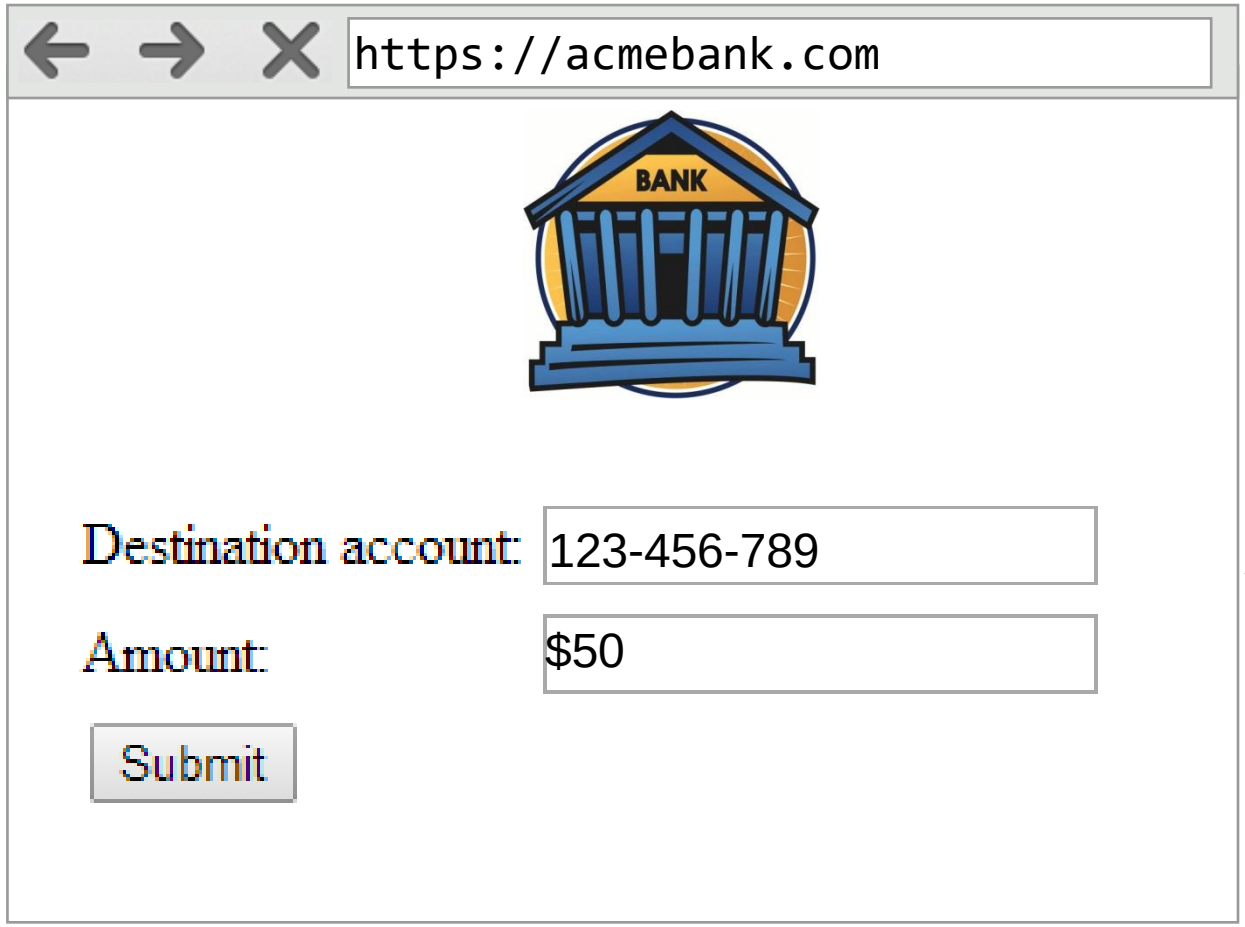
Preventing CSRF: Origin Header Checking

- Privacy-friendly version of Referer
  - Contains only the origin, not the complete URL
- Always sent along XMLHttpRequests and WebSockets
  - requires changing program logic to use these requests for state-changing operations
- In **modern browsers**, also sent along with any **cross-origin POST requests**
  - server should not necessarily rely on only having modern clients, though

What the third-party website receives

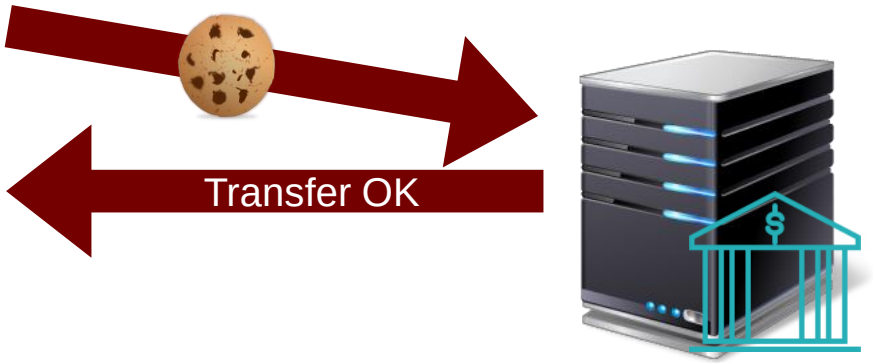
Mechanism	Sent URL
Referer	https://www.news.com/blahblah?foo=bar
Origin	https://www.news.com

Regular Web site usage



Behind the scenes

```
<form method="POST"
target=https://acmebank.com/transfer>
  <input type="text" name="acct-to">
  <input type="text" name="amount">
 <input type="hidden" name="tk" value="n73gn9ia345ntu"
  <input type="submit">
</form>
```

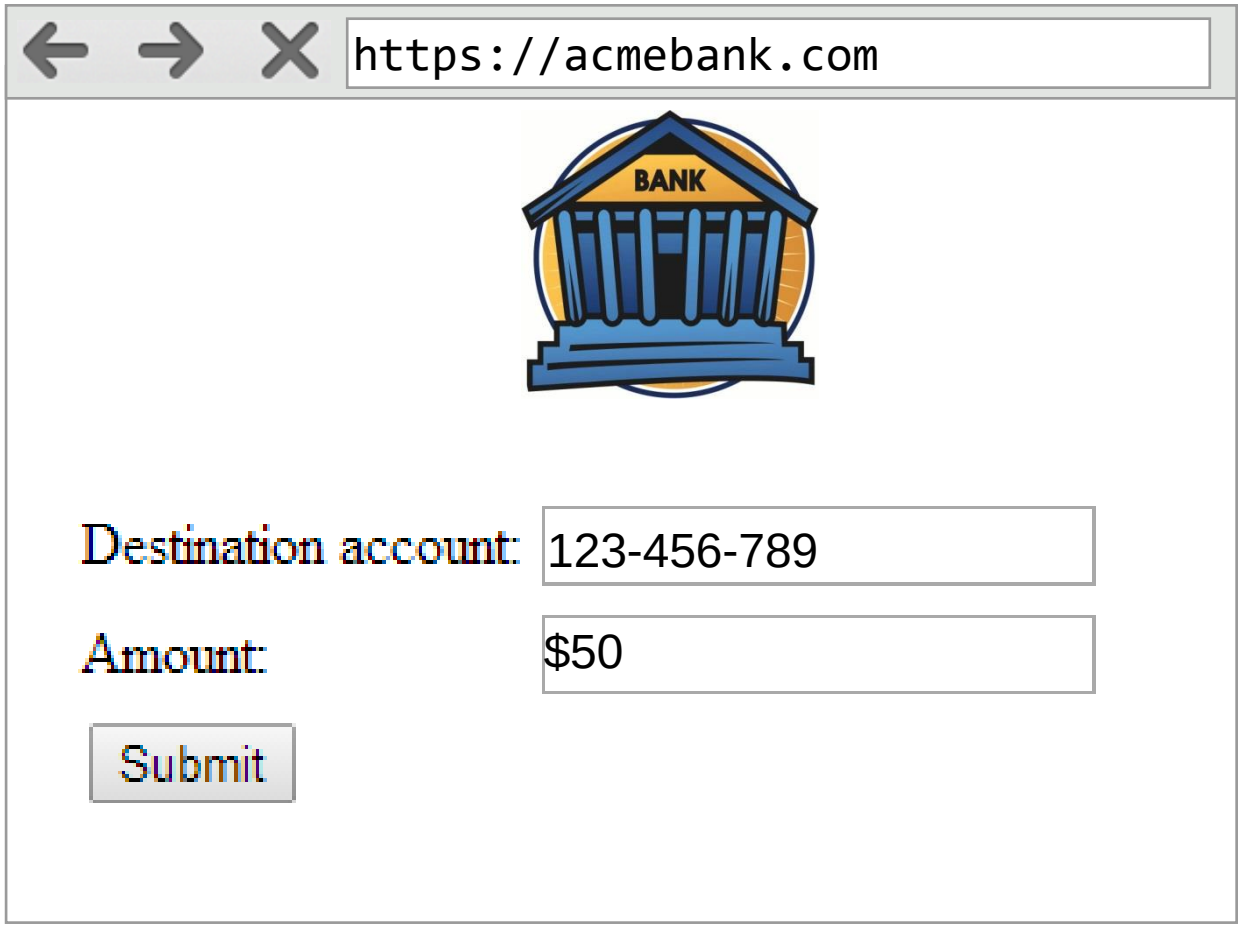


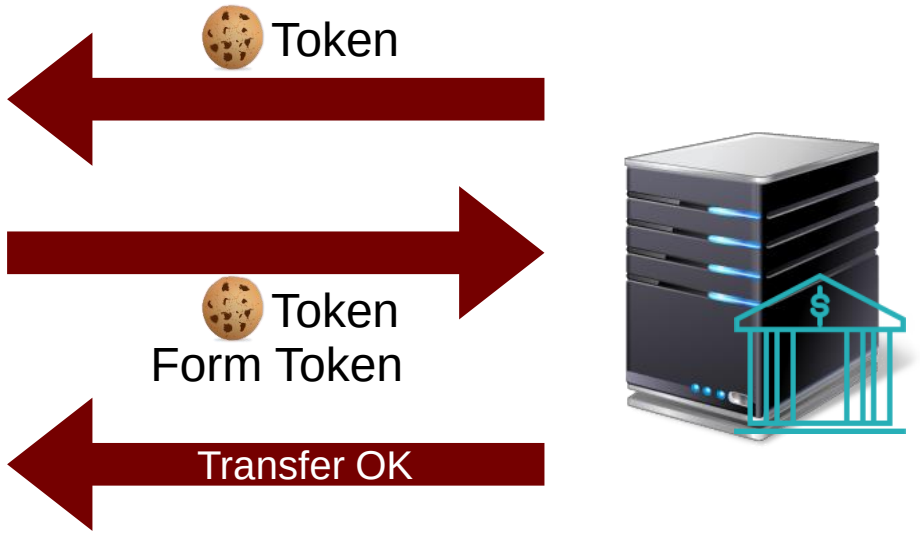
Preventing CSRF: Using CSRF tokens/nonces

Preventing CSRF: Using CSRF tokens/nonces

- Server generates token randomly for user
  - stores currently valid token in session for user
- Tokens are placed in all forms
  - inaccessible to the attacker without an XSS due to the SOP
- On submission, checks server-side token against submitted token
  - only allows action if tokens match
- Assures that a request's origin must be in the same origin

Preventing CSRF: Double Submit Cookie





Preventing CSRF: Double Submit Cookie

- Require value in posted content to match value of certain cookie
  - generate token randomly on server, store in cookie
  - insert cookie's value into each form
    - server-side addition for protected forms or
    - via JavaScript after form was loaded
- Advantage: no server-side state required
  - just compare submitted form value against cookie
- Disadvantage: cookie tossing
  - If an attacker controls a subdomain, he might set token value
  - if the server only compares cookie and form token, CSRF protection is bypassed

Preventing CSRF: Custom Headers

- Idea: use XMLHttpRequests for all state-changing requests
  - and attach a custom header (e.g., "X-CSRF-Free")
  - only handle requests with that header on the server
- Protection by existing technologies
  - Same-domain requests are always allowed
  - Cross-domain requests with custom headers requires pre-flight CORS request
- Advantage: no server-side state or randomness required
- Disadvantage: applications must be changed

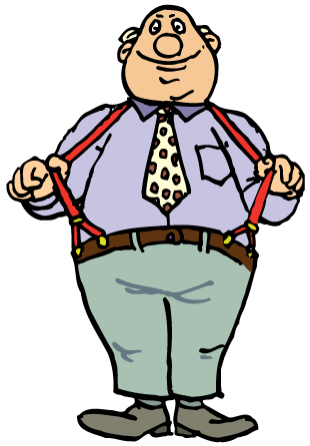
Preventing CSRF: Same-Site Cookies

- Two modes
  - Strict: even in top-level navigation, never send cookies with cross-origin request
    - if facebook.com set that, every user following a link there would not be logged in
  - Lax: non top-level navigation will not send cookies
    - cookies only send along with safe requests (GET, HEAD, OPTIONS, TRACE)
    - protects against POST-based CSRF, not against GET-based though
- Until May 2018 only supported by Chrome and Opera
- Since Chrome 80, defaults to SameSite=lax
  - SameSite=none only works with Secure flag

CSRF Conclusion

- CSRF caused by servers accepting requests from outside their origin
  - hard to determine based on Referer header though
- CSRF can have severe effects
  - compromised firmware, hijacked Web sites, ...
- Several options for fixing exist
  - CSRF tokens nowadays implemented in any (good) framework
  - protection can be achieved using well-established principles (SOP, CORS)
  - SameSite cookies also address the issue, already default in Chrome

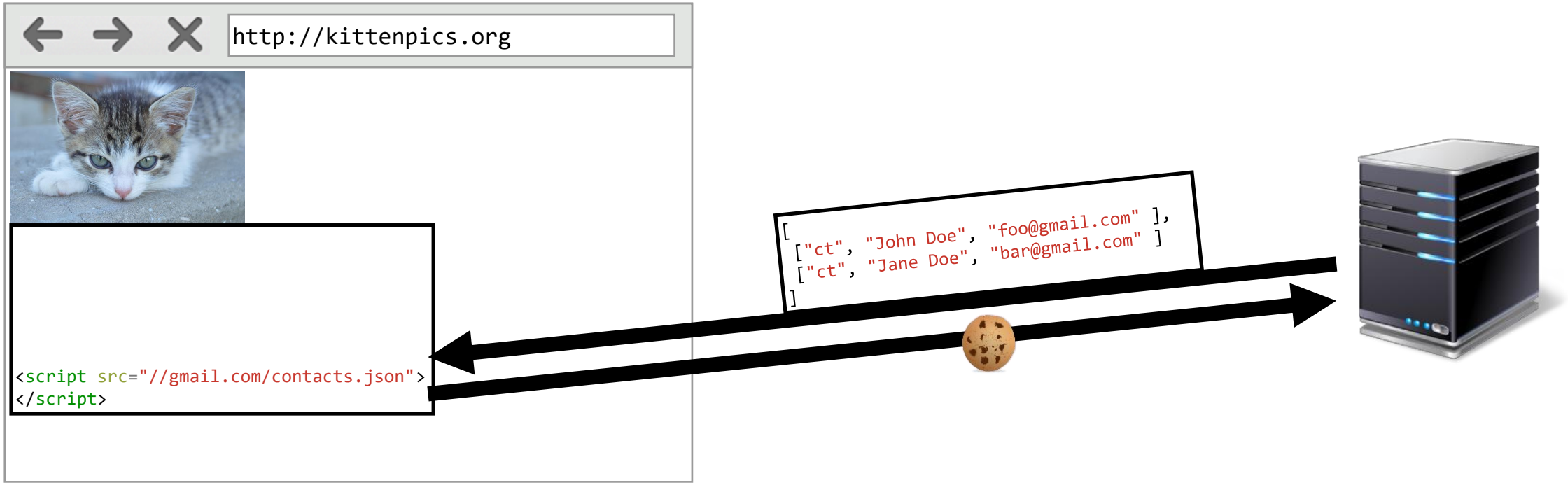
- Support still varies (https://caniuse.com/?search=samesite)
  - Use defense in depth



Cross-Origin Data Leakage

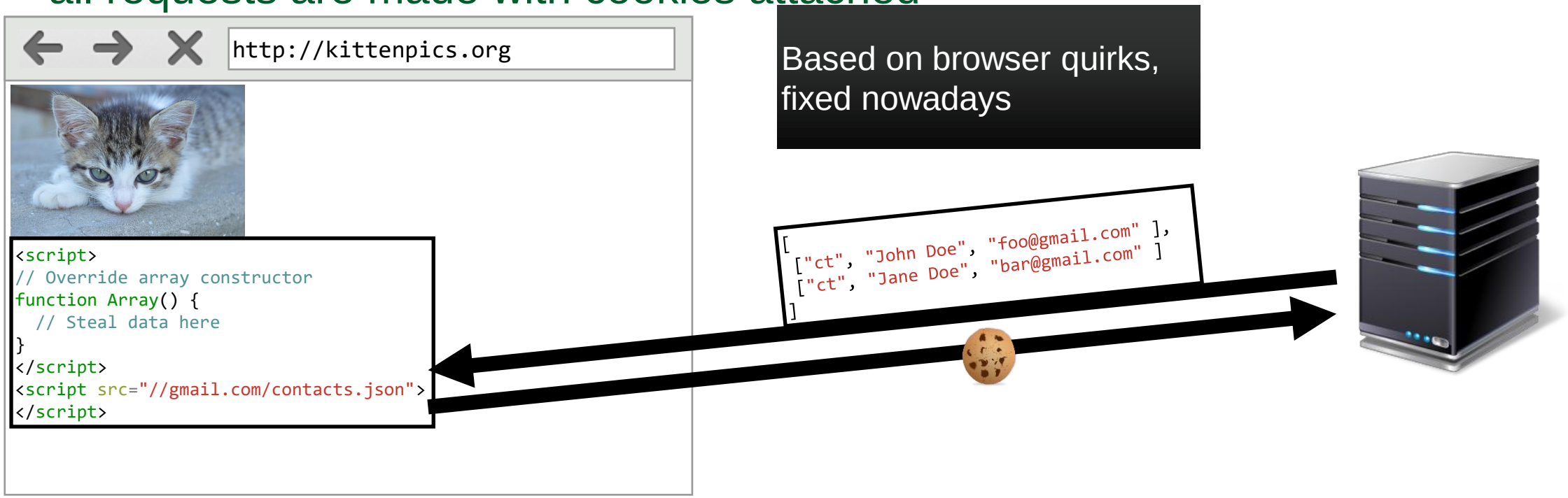
JSON/JavaScript Hijacking (2006)

- Recall from previous lectures
  - script inclusion is exempt from SOP
  - all requests are made with cookies attached



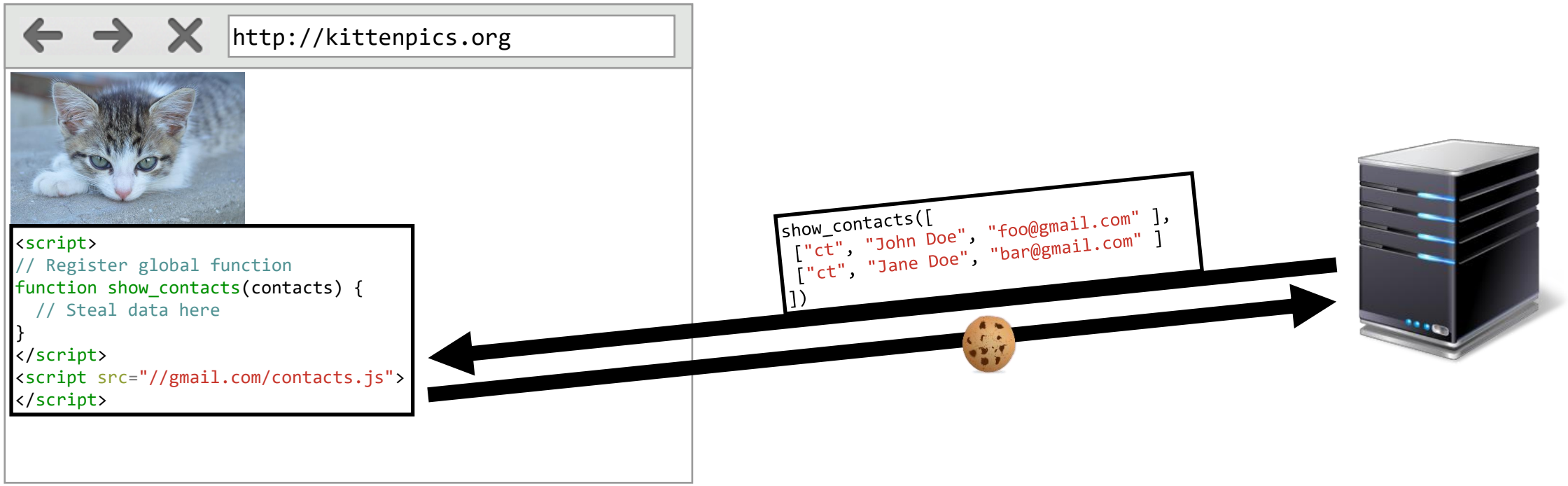
JSON/JavaScript Hijacking (2006)

- Recall from previous lectures
  - script inclusion is exempt from SOP
  - all requests are made with cookies attached



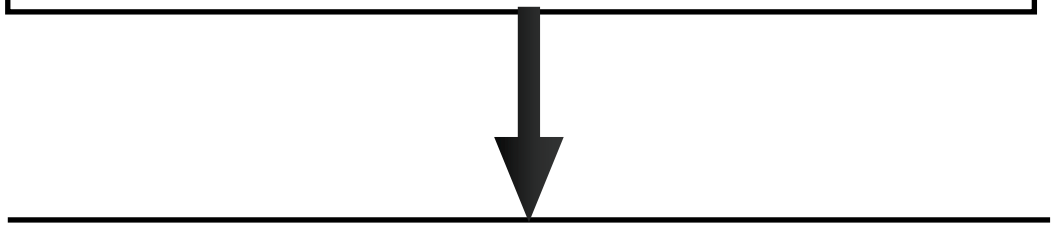
Cross-Site Scripting Inclusion (XSSI)

- Regular scripts may also be dynamically generated
  - We cannot read the source code, but can observe side-effects



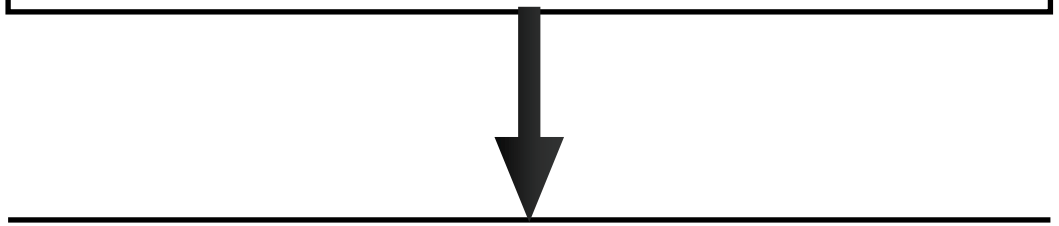
Exploiting XSSI

```
// Local variable at top level
var first_name = "John";
// Global variable due to missing var keyword
last_name = "Doe";
// Explicitly defined global variable
window.user_email = "john@doe.com";
```



```
console.log(first_name);
console.log(last_name);
console.log(user_email);
```

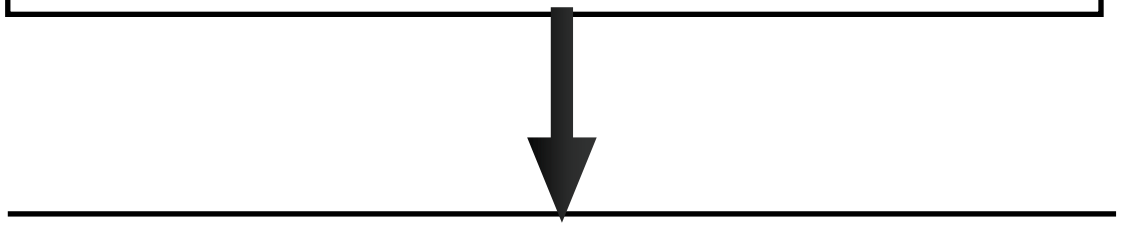
```
function example() {
  var email = "john@doe.com";
  window.MyLibrary.doSomething(email);
}

example();
```



```
window.MyLibrary = {};
window.MyLibrary.doSomething =
function(email) { console.log(email); }
```

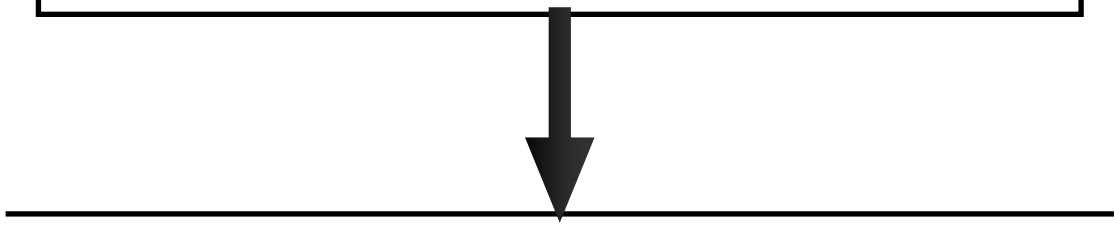
Exploiting XSSI

```
function example2() {
  var secret_values = ["secret", "more secret"];

  secret_values.forEach(function(secret) {
    // do something secret in here
  });
}
example2();
```



```
Array.prototype.forEach = function(callback) {
  // "this" is bound secret_values
  console.log(this);
}
```

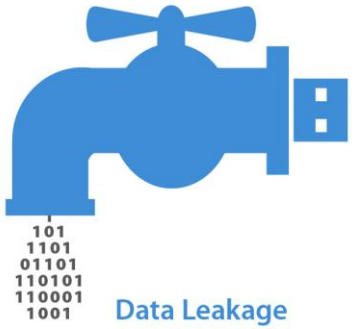
```
(function() {
  function test(someInput) {
    var email = "john@doe.com";
    doNothingWithEmail(someInput);
  }

  test.call(someThing, "myInput");
})();
```



```
Function.prototype.call = function() {
  // "this" is bound test
  console.log(this.toString());
};
```

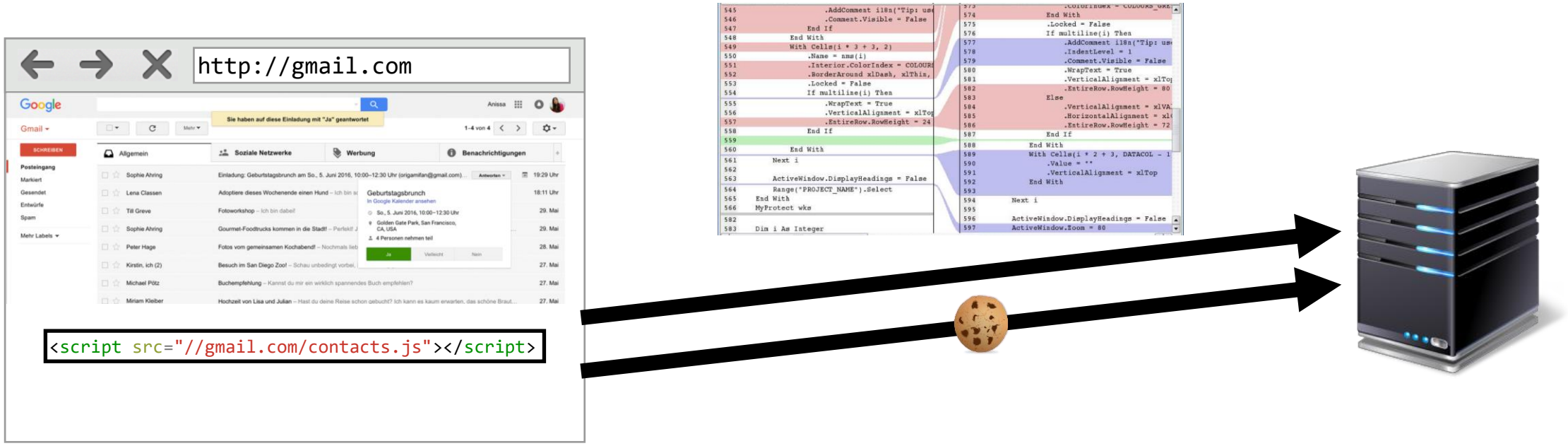
Exploiting XSSI

- Trivial case: global variables registered
  - simply access the variable (registered in global scope of site)
- Little more involved: global function called
  - overwrite function (if necessary, create object before)
- Local variables accessible if functions are called on them
  - overwrite prototype
  - e.g., forEach or call



Identifying potential XSSI [USENIX15]

- On each page visit, request included scripts twice
  - with and without cookies
- Diff the two results

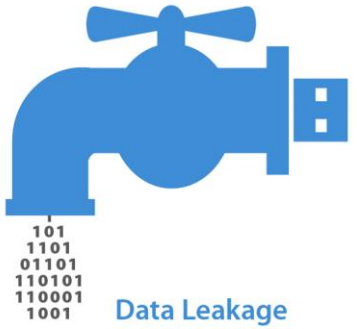


XSSI in the Wild [USENIX15]

- Conducted a study of 150 highest-ranked sites with logins
  - sites for which we could create a login (not banks, for example)

	Domains	Exploitable
Dynamic scripts	49	40
Unique identifier	34	28
Other personal data	15	11
CSRF / auth tokens	7	4

- Several high impact flaws
  - leaked credit card info on my own bank
  - reading senders and subjects of emails
  - account hijacking for file hosting service



Preventing XSSI

- Scripts must not be loadable from other origins
  - referrer checking (recall the problems associated with that)
  - use of secret tokens (similar to CSRF)
- Only provide code in scripts, use provisioning service for data
  - use XHR to retrieve data
  - easily protectable by SOP or CORS
- Use inline scripts only
  - with CSP nonces, even possible to use with CSP
  - can not be included remotely, hence data is secure there

The Great Cannon

Including third-party resources on the Web

http://cnn.com

```
<html>
....
<script src="//googletagmanager.com/tag.js">
</script>
...
</html>



var tags = "cnn.com";
document.write("Doing tagging stuff here");
// ...
```

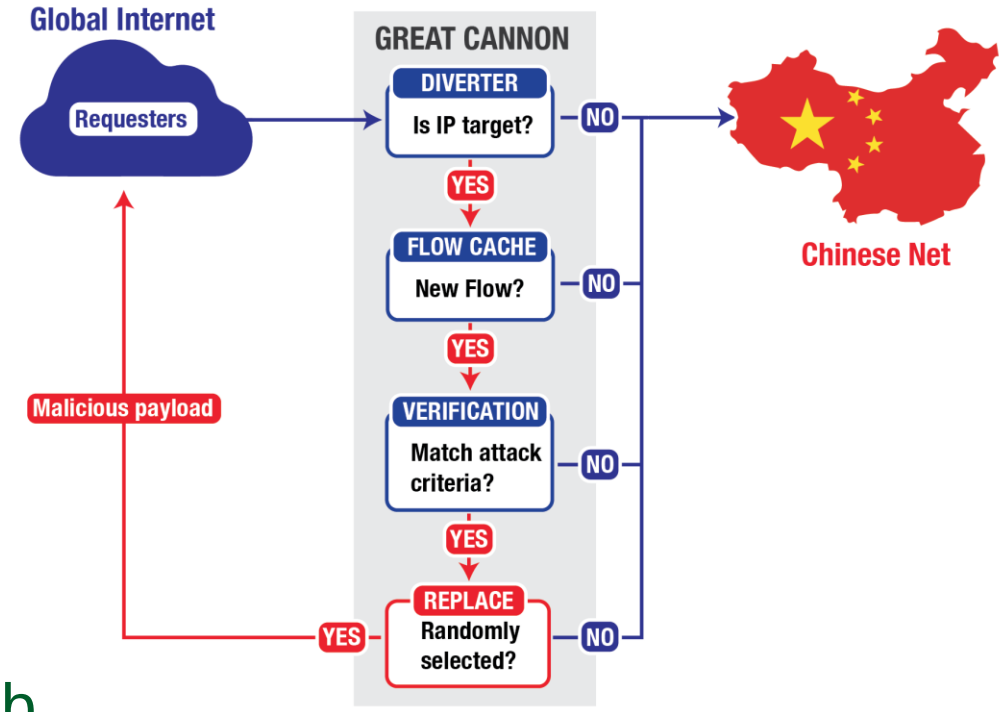
Including third-party resources on the Web (with MitM)

http://cnn.com

```
<html>
....
<script src="//googletagmanager.com/tag.js">
</script>
...
</html>


var target = "http://github.com"
var x = new XMLHttpRequest();
x.open("GET", target);
// ...
```

The Great Cannon

- China already has a powerful firewall
  - "The Great Firewall"
  - drops unwanted connections (e.g. NY Times)
- Mirror sites exists for blocked sites
  - e.g., greatfire.org and several GitHub repos
- Great Cannon injected JavaScript into content from, e.g., baidu.com
  - millions of users opened connections to GitHub, New York Times, greatfire.org
- Massively Distributed Denial of Service



https://citizenlab.ca/2015/04/chinas-great-cannon/

Subresource Integrity (SRI)

- To thwart such injection attacks, SRI was proposed
- Use cryptographic hash of remote resource
  - for scripts and style sheets
  - if hash does not match, resource is ignored

```
<script src="https://code.jquery.com/jquery-2.1.4.min.js"
integrity="sha384-R4/ztc4ZlRqWjqIuvf6RX5yb/v90qNGx6fS48N0tRxiGkqveZETq72KgDVJCp2TC"
crossorigin="anonymous"></script>
```

- Protects against malicious CDNs/MitM attackers
  - also allows to pin to a specific version of third-party libraries

```
<script>window.jQuery || /* reload from own domain here */;</script>
```

Subresource Integrity (SRI)

- SRI resources must be CORS-enabled
  - otherwise, SRI could be used to test remote resource for certain content
- Integrity attribute can have multiple values
  - Only strongest hash is used

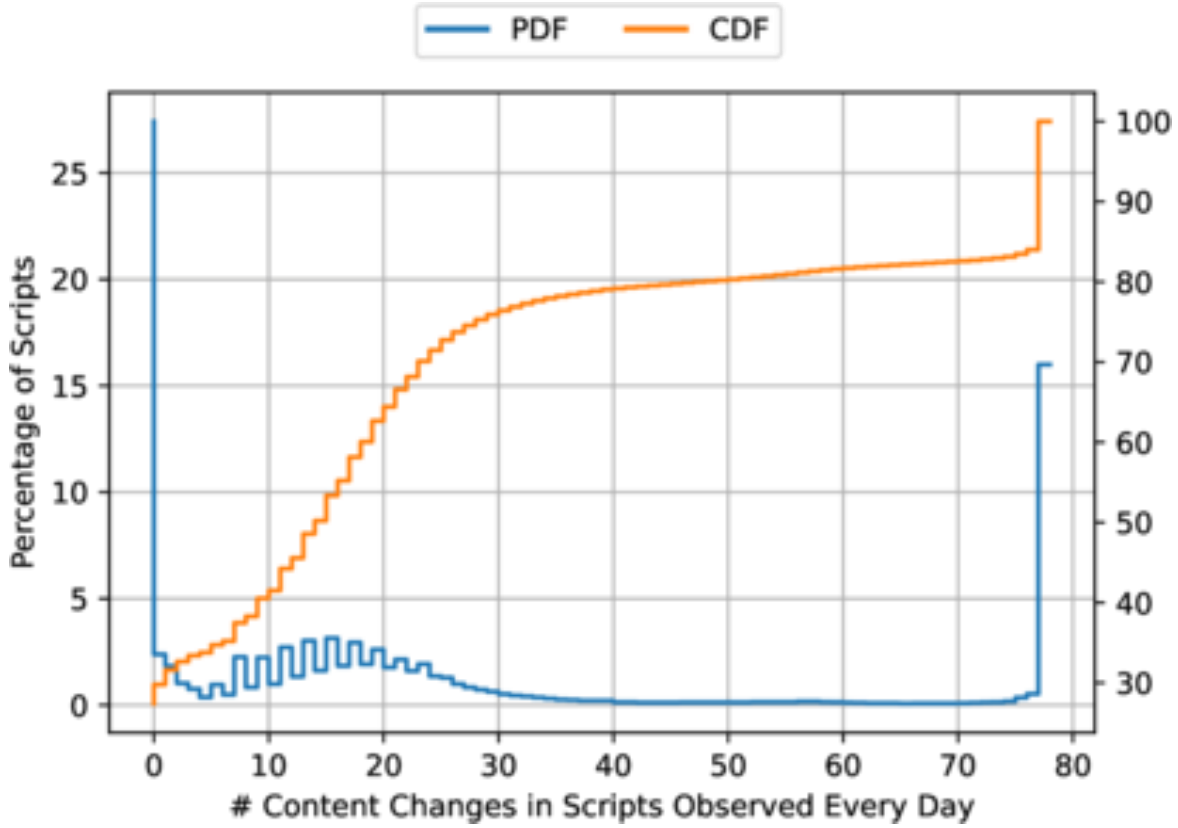
```
<script src="https://code.jquery.com/jquery-2.1.4.min.js"
integrity="sha384-R4/ztc4ZlRqWjqIuvf6RX5yb/v90qNGx6fS48N0tRxiGkqveZETq72KgDVJCp2TC sha256-
8WqyJLuWKRBVhxXIL1jBDD7SDxU936oZkCnxQbWwJVw="
crossorigin="anonymous"></script>
```

  - Multiple same-strength hashes are allowed but rarely used

```
<script src="https://code.jquery.com/jquery-2.1.4.min.js"
integrity="sha256-t1X5SBfMY4/0kYdt8H1CP/90GgOi1G6U9UnjC6AVYHA=
sha256-8WqyJLuWKRBVhxXIL1jBDD7SDxU936oZkCnxQbWwJVw="
crossorigin="anonymous"></script>
```

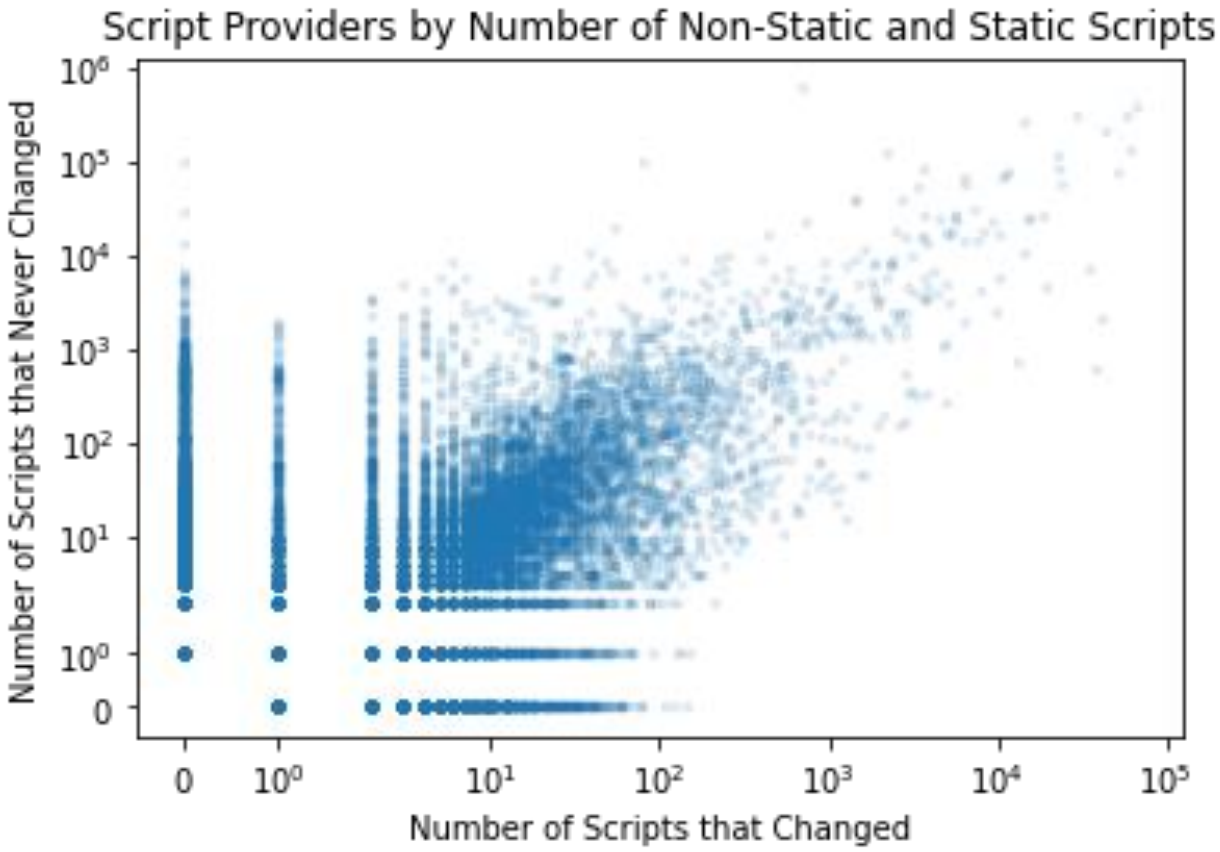
Subresource Integrity (SRI) [WWW'23]

- SRI is applicable to static scripts, or scripts that rarely change
- What is the fraction of scripts that remain static?
  - 27% of scripts never changed
  - 16% of scripts changed every day



Subresource Integrity (SRI) [WWW'23]

- SRI needs to be applied to every script from each third party
- How many third-party providers serve only static scripts?
  - 44% of providers serve only static scripts (x=0)
  - 24% of providers serve only changing scripts (y=0)
  - 32% of providers serve a mix of both



Sandboxing Content

Multi-origin Web applications

- Modern Web applications use code from multiple origins
  - Analytics
  - Advertisement
  - Maps
  - ....
- Even framed content may, e.g., open a popup
  - or redirect the parent frame
- Necessity for control privileges of included content arises
  - putting everybody in their own little sandbox

Sandboxing iframes

- Limits iframe's ability to conduct certain actions
  - e.g., disable JavaScript, putting them in an isolated origin
- Just adding sandbox to the iframe will restrict everything
  - rights have to be granted explicitly
  - `allow-forms`: allows for form submission in iframe
  - `allow-popups`: enables popups
  - `allow-pointer-lock`: enable PointerLock API to get raw mouse movements
  - `allow-scripts`: enable scripting
  - `allow-same-origin`: enable origin of included page, not isolated one
  - `allow-top-navigation`: enables navigating the top frame

Sandbox usage examples

```
<textarea id='code'></textarea>
<button id='safe'>eval() in a sandboxed frame.</button>
<iframe sandbox='allow-scripts' id='sbox' src='frame.html'>
</iframe>

<script>
  function evaluate() {
    sandboxed.contentWindow.postMessage(code.value, '*');
  }
  safe.addEventListener('click', evaluate);

  window.addEventListener('message', function (e) {
    if (e.origin === "null" && e.source === sbox.contentWindow)
      alert('Result: ' + e.data);
    });
</script>
```

Parent page

```
<script>
  window.addEventListener('message', function (e) {
    if (e.origin !== "https://main.com") {
      return
    }
    var mainWindow = e.source;
    var result = '';

    try {
      result = eval(e.data);
    } catch (e) {
      result = 'eval() threw an exception.';
    }
    mainWindow.postMessage(result, e.origin);
  });
</script>
```

frame.html

https://www.html5rocks.com/static/demos/evalbox/index.html

Determining least privilege

- Example: tweet button
  - opens popup window
  - submit a form
  - sends authenticated request to twitter.com (using and accesses document.cookie)
- Requires four permissions
  - allow-popups (well, it opens a popup..)
  - allow-forms (well, it is a form)
  - allow-same-origin (JavaScript needs access to cookies)
  - allow-scripts (not too much of a surprise)

Determining least privilege

- Example: tweet button
  - opens popup window
  - submit a form
  - sends authenticated request to twitter.com (using and accesses document.cookie)
- Requires fo
  - allow-popups (well, it opens a popup..)
  - allow-forms (well, it is a form)
  - allow-same-origin (JavaScript needs access to cookies)
  - allow-scripts (not too much of a surprise)

```
<iframe sandbox="allow-same-origin allow-scripts allow-popups allow-forms"
    src="https://platform.twitter.com/widgets/tweet_button.html"
    style="border: 0; width:130px; height:20px;"></iframe>
```

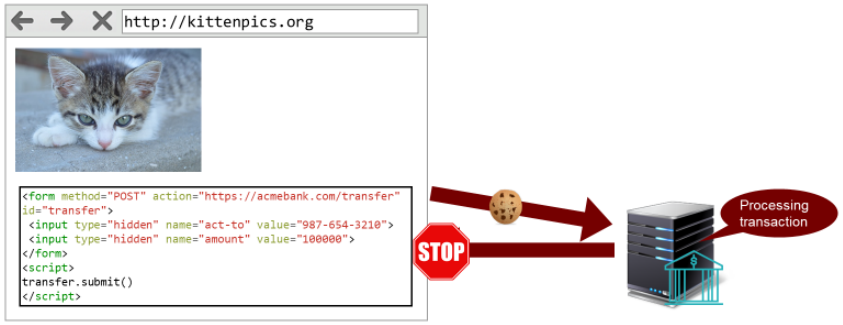
Sandboxing with CSP

- Limits the entire page's ability to conduct certain actions
  - as if the page were loaded in an iframe with the sandbox attribute
- What's the difference?
  - Functionally the same, but they differ in scope
  - CSP sandboxing applies to the entire page
  - iframe sandboxing applies only to that iframe
- When would you use one over the other?
  - e.g., using CSP sandboxing on a page that is an archived instance of a web vulnerability that triggers drive-by downloads

Summary

4

Forcing browser to perform an action for the attacker



35

Subresource Integrity (SRI)

- To thwart such injection attacks, SRI was proposed
- Use cryptographic hash of remote resource
  - for scripts and style sheets
  - if hash does not match, resource is ignored

```
<script src="https://code.jquery.com/jquery-2.1.4.min.js"
integrity="sha384-R4/ztc4ZlRqWjqIuvf6RX5yb/v90qNGx6fS48N0tRxiGkqveZETq72KgDVJCp2TC"
crossorigin="anonymous"></script>
```

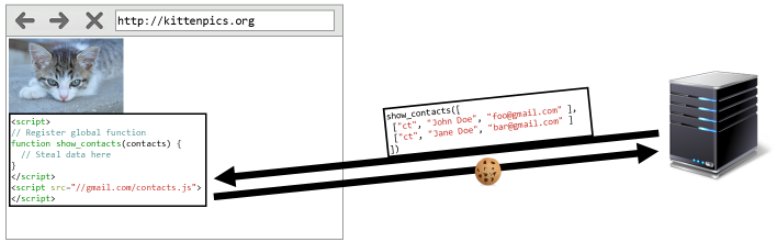
- Protects against malicious CDNs/MitM attackers
  - also allows to pin to a specific version of third-party libraries

```
<script>window.jQuery || /* reload from own domain here */;</script>
```

24

Cross-Site Scripting Inclusion (XSSI)

- Regular scripts may also be dynamically generated
  - We cannot read the source code, but can observe side-effects



40

Sandbox usage examples

```
<textarea id='code'></textarea>
<button id='safe'>eval() in a sandboxed frame.</button>
<iframe sandbox='allow-scripts' id='sbox' src='frame.html'>
</iframe>

<script>
  function evaluate() {
    sandboxed.contentWindow.postMessage(code.value, '*');
  }
  safe.addEventListener('click', evaluate);

  window.addEventListener('message', function (e) {
    if (e.origin === "null" && e.source === sbox.contentWindow)
      alert('Result: ' + e.data);
    });
</script>
```

Parent page

```
<script>
  window.addEventListener('message', function (e) {
    if (e.origin !== "https://main.com") {
      return
    }
    var mainWindow = e.source;
    var result = '';

    try {
      result = eval(e.data);
    } catch (e) {
      result = 'eval() threw an exception.';
    }
    mainWindow.postMessage(result, e.origin);
  });
</script>
```

frame.html

https://www.html5rocks.com/static/demos/evalbox/index.html

Credits

- Original slide deck by Ben Stock
- Modified by Nick Nikiforakis