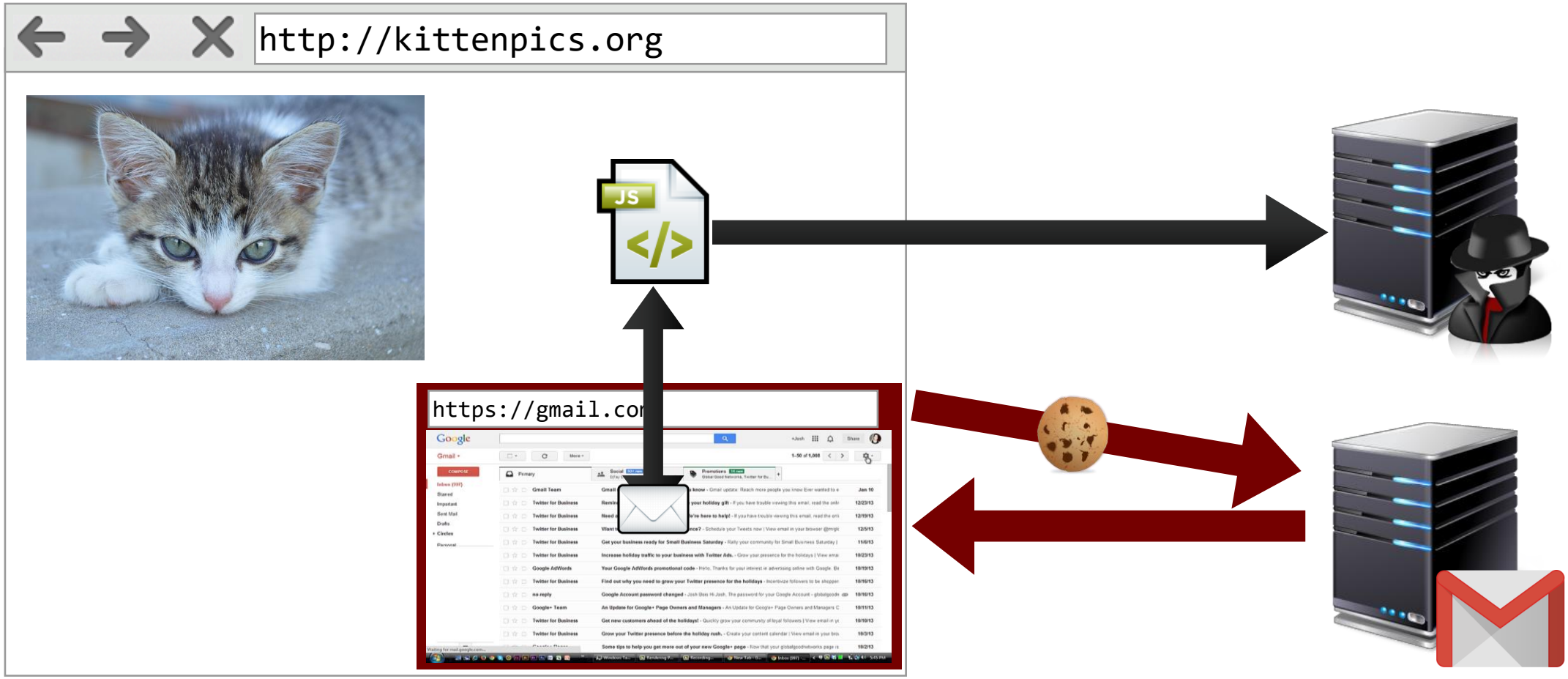
Stony Brook University

CSE 361: Web Security

Cross-domain Communication

Nick Nikiforakis

A World Without Separation between Sites

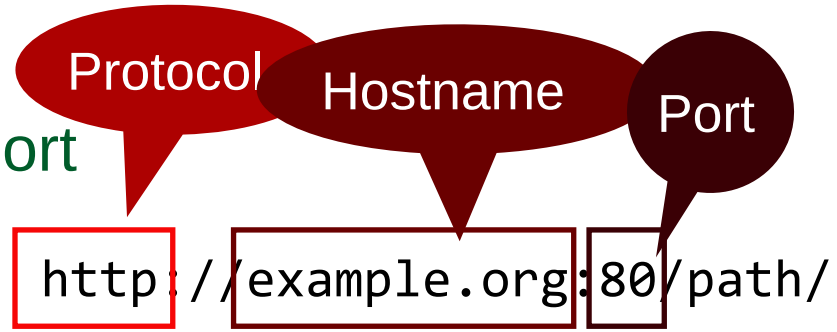


The Same-Origin Policy for JavaScript

- Most basic access control policy
  - controls how active content can access resources
- Same-Origin Policy for JavaScript for three actions
  - Script access to other document in same browser
    - frames/iframes
    - (popup) windows
  - Script access to application-specific local state
    - cookies, Web Storage, or IndexedDB
  - Explicit HTTP requests to other hosts
    - XMLHttpRequest

The Same-Origin Policy for JavaScript

- Only allows access if origins match
  - Origin defined by protocol, hostname, and port

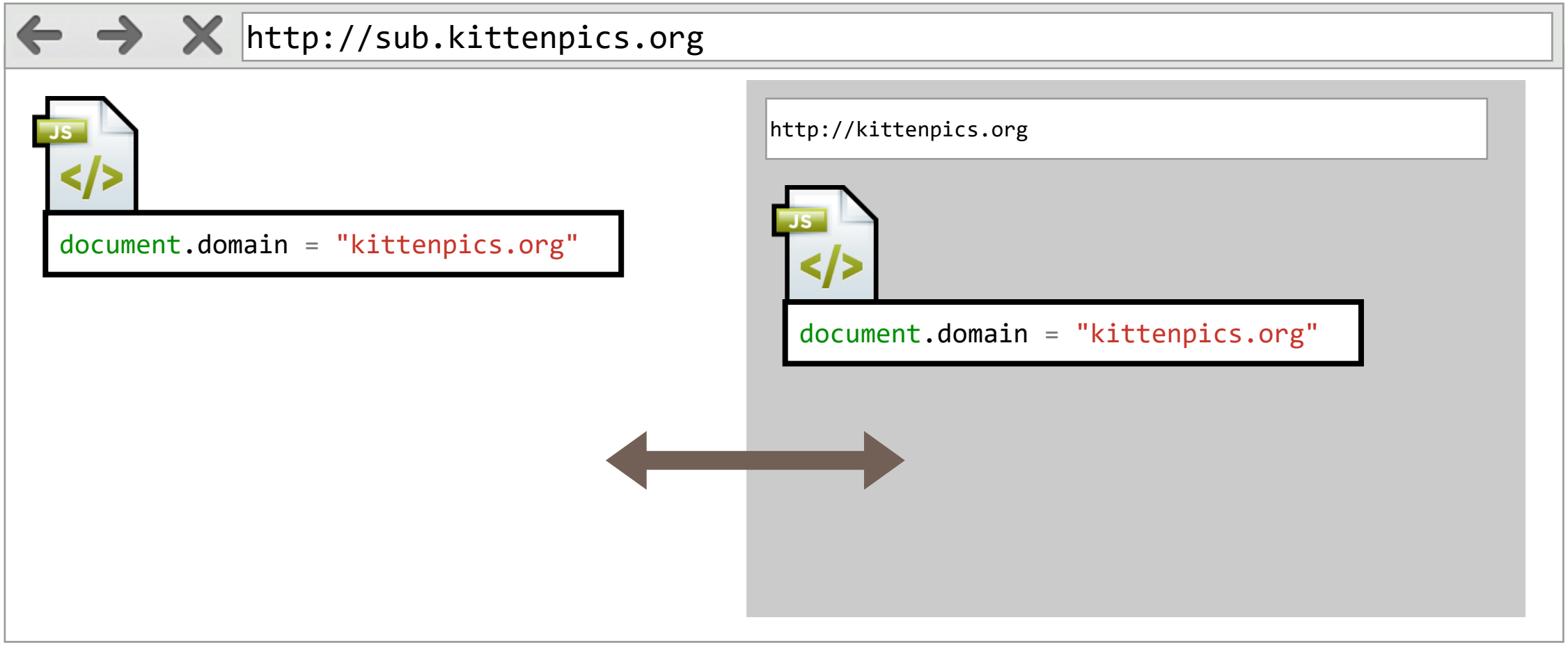


Originating document	Accessed document	Non-IE Browser	Internet Explorer
http://example.org/a	http://example.org/b	✓	✓
http://example.org	http://www.example.org	⊘	⊘
http://example.org	https://example.org	⊘	⊘
http://example.org	http://example.org:81	⊘	✓

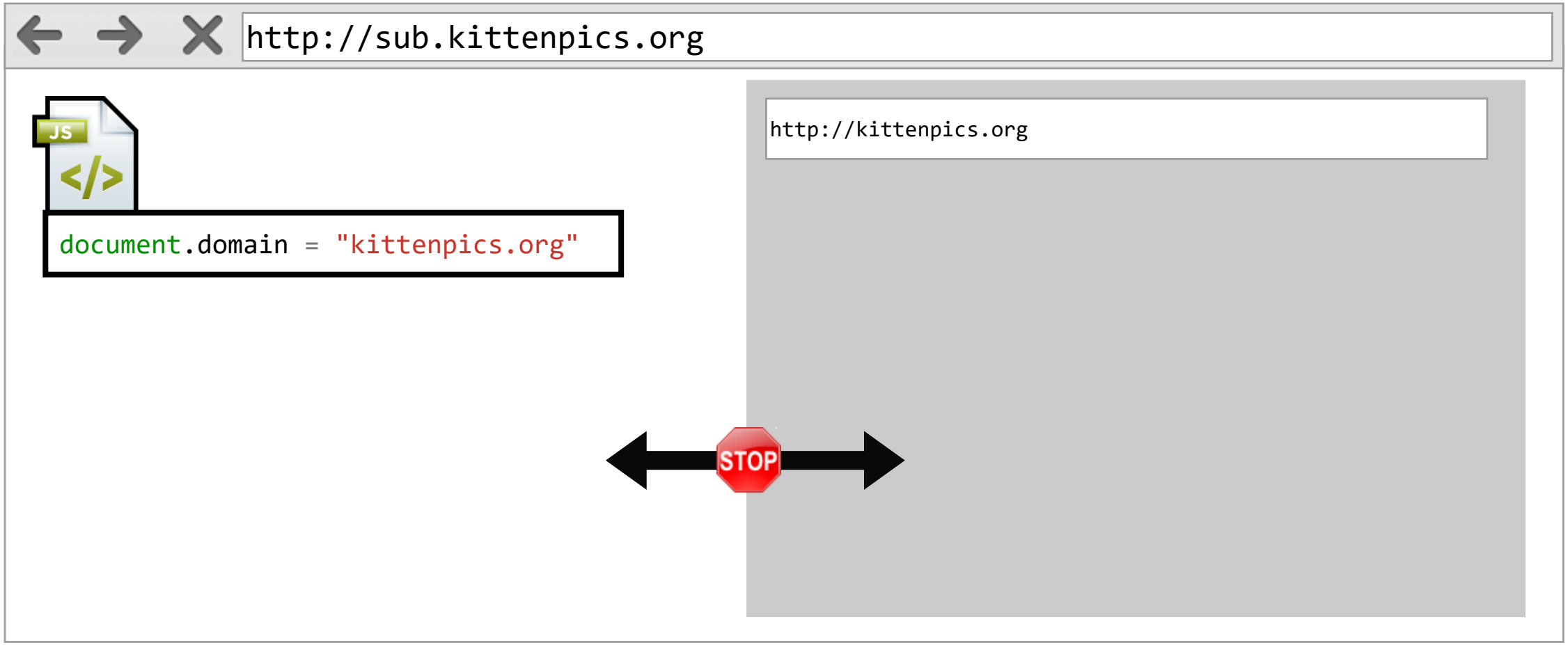
Domain Relaxation

- Two sub-domains of a common parent domain want to communicate
  - Notably: can overwrite different port!
- Browsers allow setting `document.domain` property
  - Can only be set to valid suffix including parent domain
  - `test.example.org -> example.org` ok
  - `example.org -> org` forbidden
- When first introduced, relaxation of single sub-domain was sufficient
- Nowadays: both (sub-)domains must explicitly set `document.domain`

Domain Relaxation



Domain Relaxation



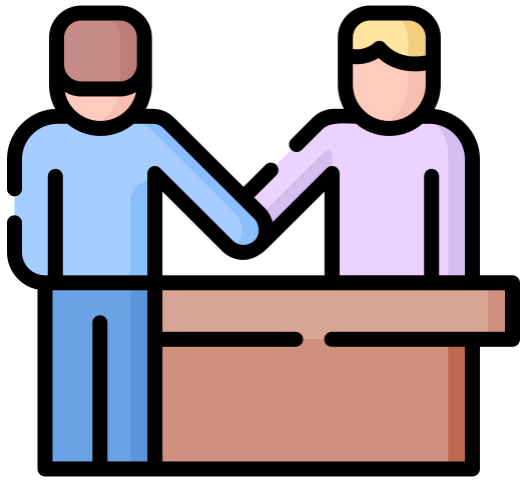
Cross-Origin Communication

Cross-origin communication

- Subdomains of the same domain can use domain relaxation when they want to talk to one another.

- What happens when two different domains want to be able to communicate?
  - E.g. Access your Dropbox files via your Gmail account?

- No way to “relax” to a different domain
- Same Origin Policy will block the access
  - JavaScript can be used to automatically submit an HTML form but cannot read the response



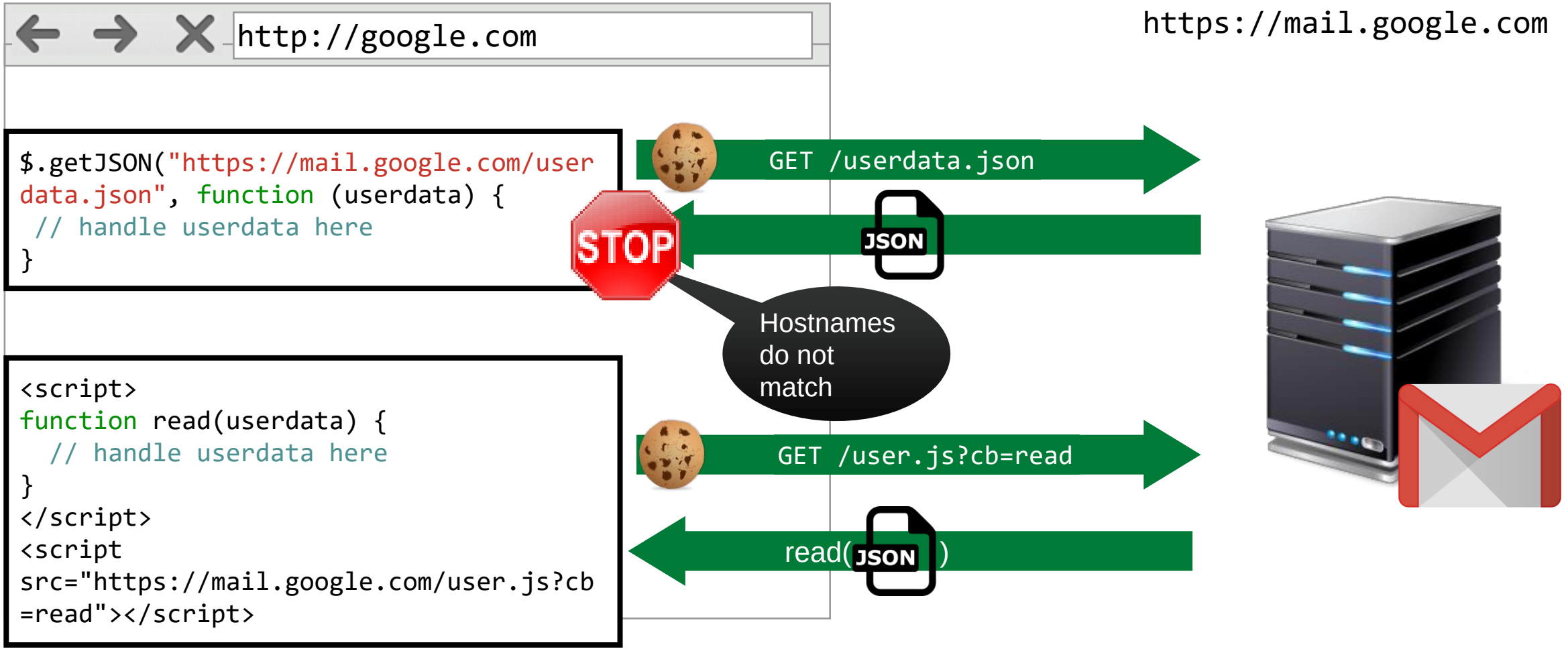
Cross-Domain Communication: window.name

- SOP prohibits access to other document's window object
  - unless origins match/are relaxed
- On the Web, there is always an exception to the rule
  - in this case, the name of a window
- Dirty Hack
  - load "communication partner" in frame
  - have other site set window.name
  - access frame.name
- Should not really be used for anything new

Cross-Domain Communication: JSONP

- Recall Web model: may include resources from remote origins
  - access from JavaScript to cross-domain resources is restricted though
- Weird case: scripts
  - can be included from remote origin
  - execute in **including** origin (side effects observable on global scope)
  - source code not accessible from including page
- JSONP ("JSON with Padding") (ab)uses this
  - callback function as parameter
  - creates script code dynamically

JSONP Concept



The Rosetta Flash Attack

- Discovered in 2014 by Michele Spagnuolo of Google
- **Flash does not inherit origin when included**
  - Flash on `a.com` can conduct GET and POST requests (with cookies) to `a.com`
  - even if it is included in `b.com`
- JSONP allows attacker to control first bytes of a file (callback function)
  - albeit most likely only ASCII characters
- Flash will execute any file as SWF if
  - object tag with correct Content-Type is used
  - file appears to be valid flash

The Rosetta Flash Attack

- Requirement for big trouble: find ASCII-only Flash file
  - possible by utilizing zlib and Huffman encoding in just the right way
  - See https://miki.it/blog/2014/7/8/abusing-jsonp-with-rosetta-flash/ for technical details
- Exploit by setting callback parameter accordingly
  - `<object type="application/x-shockwave-flash" data="https://vulnerable.com/endpoint?callback=CWSMIKI0hCD0Up0IZUnnnnnnnn.."></object>`
- Affected Google, Twitter, LinkedIn, ...

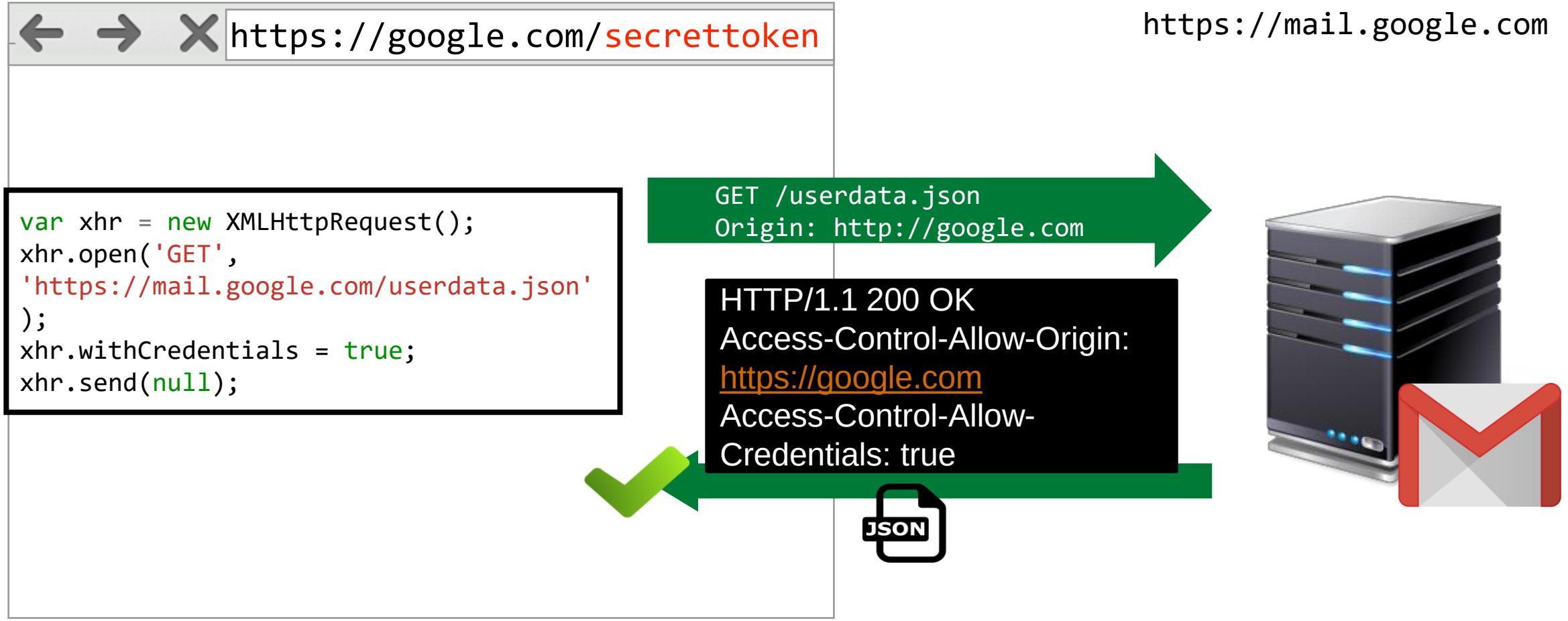
JSONP Drawbacks

- Dirty hack around the Same-Origin Policy
- Full trust on remote website
  - Website providing a JSONP response can instead provide malicious code
- No error management
  - What happens when the cross-origin JSONP request fails (e.g. 404 response)?
- Only "security" measure: Referer checking
  - Referer can be stripped
    - on purpose by **attacker**
    - for privacy reasons by **proxies**
- Callback controllable by attacker
  - partially controllable resource on JSONP-hosting server

Cross-Domain Communication: CORS

- Goal: enable fine-grained cross-domain access
- Result: Cross-Origin Resource Sharing
  - Policy sent by server, interpreted by browser accordingly
  - Uses **Origin** header (not complete URL)
- HTTP headers sent along with response
  - for **simple** requests, headers interpreted by browser afterwards

CORS Concept (simple request)



CORS Response Headers (part 1)

- `Access-Control-Allow-Origin: <origin> | *`
  - Controls whether origin can access
  - * allows all, but cannot be used with credentialed requests
  - **Secure by default**

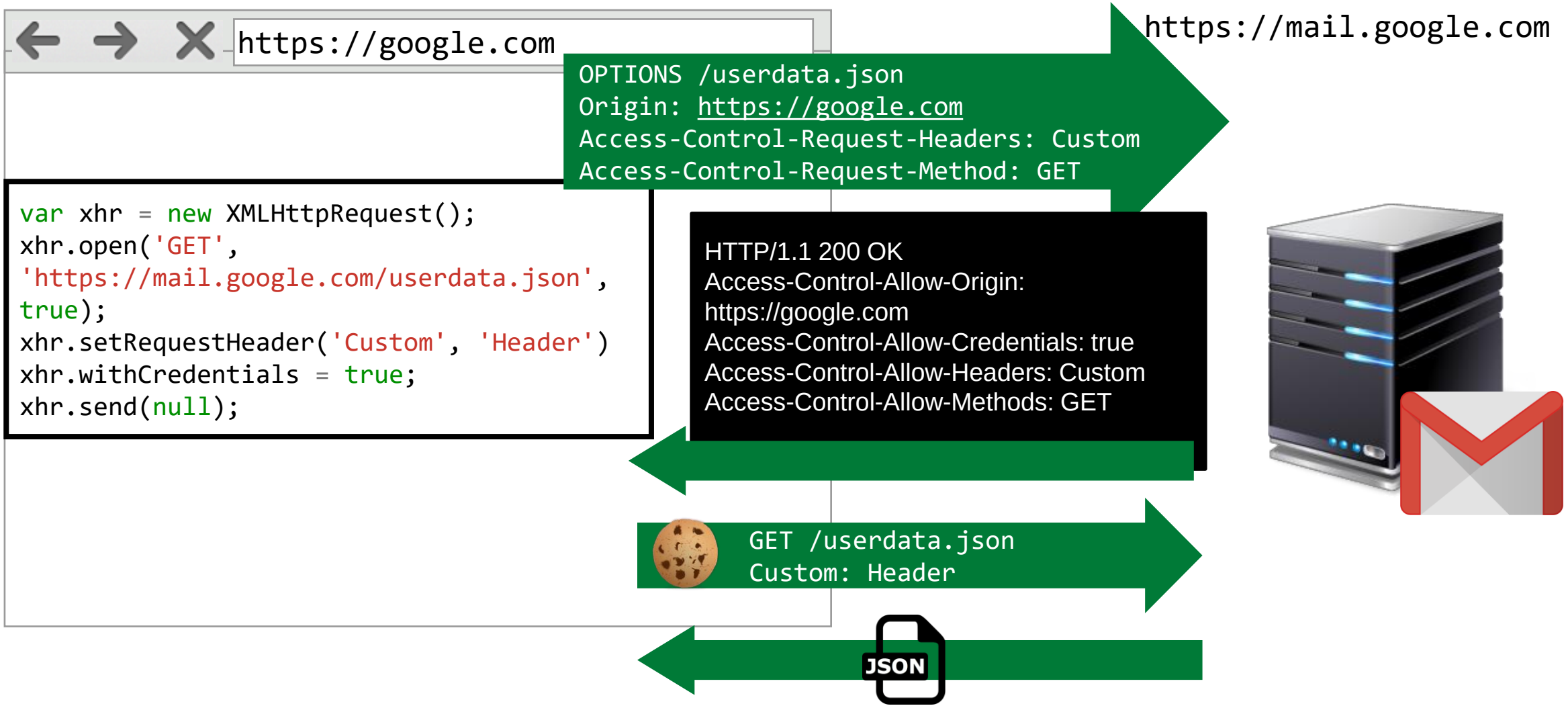
- `Access-Control-Allow-Credentials`
  - Controls whether response can also be read if credentials (mostly cookies) were requested

- `Access-Control-Expose-Headers`
  - Specifies which headers may be accessed by JavaScript
  - By default, 6 simple response headers are always allowed
    - Cache-Control, Content-Language, Content-Type
    - Expires, Last-Modified, Pragma

CORS Complex requests

- CORS serves both functionality and security goals
  - Allow cross-domain sharing of data
  - Must not make requests less secure
- Idea: simple requests can be executed via JavaScript in any case
  - POST/GET requests by automatically filling a form
  - Everything that goes beyond capability of JS + forms must be protected
- Complex requests require **preflight** request
  - e.g., PUT/DELETE/CONNECT, custom headers, …
  - preflight: OPTIONS request with requested parameters

CORS Preflight requests



CORS Request Headers

- `Access-Control-Request-Method`
  - Specify which HTTP method is supposed to be used by following cross-origin request
- `Access-Control-Request-Headers`
  - List all headers that client wants to use for subsequent request
- `Origin`
  - Contains origin of the resource wanting to make a cross-origin request
  - (privacy-friendly: no full URL)

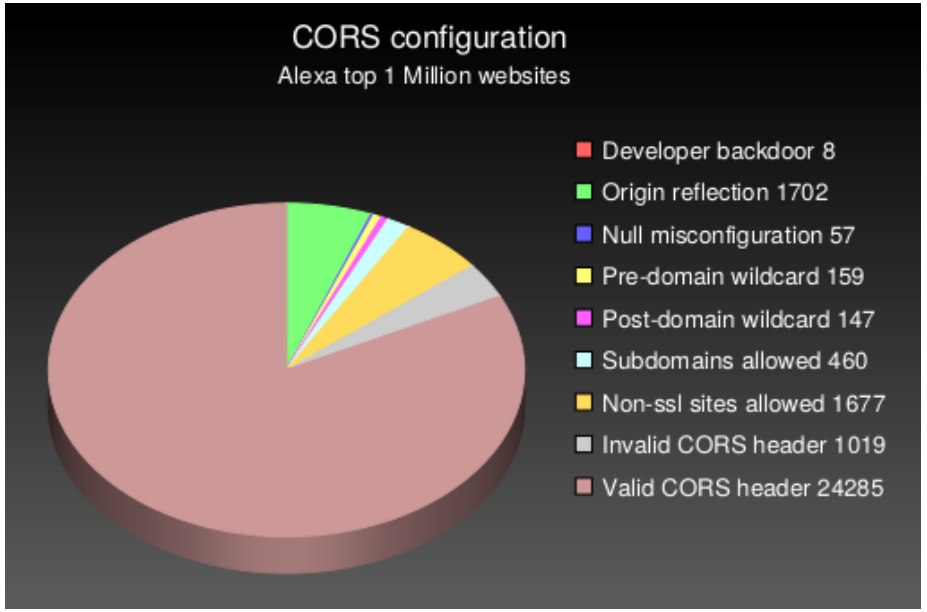
CORS Response Headers (part 2)

- `Access-Control-Max-Age`
  - Control cache lifetime of **preflight** request
- `Access-Control-Allow-Methods`
  - Determines which HTTP methods are allowed by preflighted request
- `Access-Control-Allow-Headers`
  - Determines which HTTP headers may be sent by client in preflighted request

CORS - How you can still mess up

```
function cors() {
    // Allow from any origin
    if (isset($_SERVER['HTTP_ORIGIN'])) {
        // Decide if the origin in
$_SERVER['HTTP_ORIGIN'] is one
        // you want to allow, and if so:
        header("Access-Control-Allow-
Origin: {$_SERVER['HTTP_ORIGIN']}");
        header('Access-Control-Allow-
Credentials: true');
        header('Access-Control-Max-Age:
86400');    // cache for 1 day
    }
    // ....
}
```

https://stackoverflow.com/a/9866124/1613775



http://web-in-security.blogspot.de/2017/07/cors-misconfigurations-on-large-scale.html

CORS Misconfiguration - null Origin

```
header("Access-Control-Allow-Credentials: true");

if (is_allowed($_SERVER['HTTP_ORIGIN'])) {
  header("Access-Control-Allow-Origin: {$_SERVER['HTTP_ORIGIN']}");
} else {
  header("Access-Control-Allow-Origin: null");
}
```

- Attacker can force null origin to be sent
  - data URLs have their origin set to null

```
<iframe src='data:text/html,
<html><body><script>
xhr = new XMLHttpRequest();
xhr.open("GET", "https://null-origin-cors.com/file", false);
xhr.send();
alert(xhr.responseText);
</script></body></html>'></iframe>
```

Domain Relaxation vs. CORS

- Domain relaxation affects two documents
  - JavaScript API
  - two HTML pages use document.domain to set a **common parent domain**
  - these documents can read any data and call any API in other document
- CORS allows fine-grained access control to any origin
  - HTTP header
  - no need to share parent domain
  - if domain a.com sets CORS to `Access-Control-Allow-Origin: b.com`, b.com can read the content from a.com
    - does not allow b.com to call any JavaScript in a.com
  - Requesting page sends along its **origin** in HTTP header

PostMessages

Cross-Domain Communication: Web Messaging

- Goal: enable safe message exchange between two sites
  - preserving authenticity and confidentiality of a message
- Result: HTML5 Web Messaging
  - initially only postMessage, now also supports message channels
- Messages can be sent via window.postMessage
- .. and received with an event handler

postMessage Concept


```
window.addEventListener("message",
    receiveMessage);

function receiveMessage(event)
{
  if (event.origin !== "http://main.site")
    return;
  var message = event.data;
  process(message);
}
```

postMessage security

- Origin of message can be checked
  - can ensure authenticity of message
  - message.origin provides origin
- Target can be specifically set
  - if target is not matched, message is not delivered
  - allows for wildcards

```
window.addEventListener("message",
receiveMessage);

function receiveMessage(event)
{
  var message_data = event.data;
  // we expect this to be JSON
  if (event.origin == 'http://expected.site') {
    var data = eval(message_data);
  }
}
```

```
window.addEventListener("message",
receiveMessage);

function receiveMessage(event)
{
  var message_data = event.data;
  if (message_data == 'get_userdata') {
    event.source.postMessage('....',
'http//target.site');
  }
}
```

postMessage insecurity

- Origin of message can be checked
  - can ensure authenticity of message
  - message.origin provides origin
- Target can be specifically set
  - if target is not matched, message is not delivered
  - allows for wildcards

```
window.addEventListener("message",
receiveMessage);

function receiveMessage(event)
{
  var message_data = event.data;
  // we expect this to be JSON

    var data = eval(message_data);

}
```

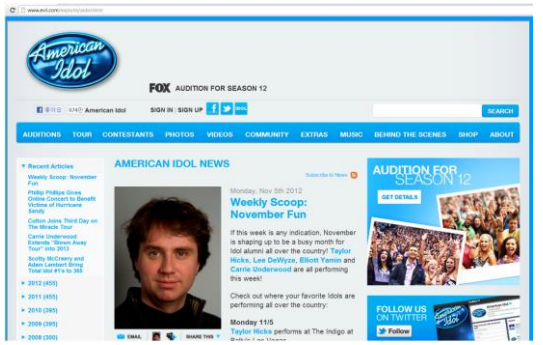
```
window.addEventListener("message",
receiveMessage);

function receiveMessage(event)
{
  var message_data = event.data;
  if (message_data == 'get_userdata') {
    event.source.postMessage('....', '*');

  }
}
```

postMessage in the Wild [NDSS 2013]

- Son and Shmatikov investigated insecure use of postMessage
- Checked Alexa Top 10,000 domains for origin checks
  - missing completely
  - improper checks
    - `if(a.origin && a.origin.match(/\.kissmetrics\.com/))`
    - `if(/id.rambler.ru$/i.test(a.origin))`
    - `a.origin.indexOf("widgets.ign.com") !== -1`
- Found 84 popular sites to be vulnerable
  - Cross-Site Scripting
  - Persistent changes to Web Storage



postMessage in the Wild [USENIX 2017]

- Recent study from CISPA about how the Web evolved
  - Analysis of the top 500 sites over 20 years

	postMessage received	no origin check	postMessage sent	wildcard target
2009	0.5%	0%	20.9%	2.2%
2010	10.8%	2.4%	5.9%	3.9%
2011	18.5%	8.4%	19.0%	14.8%
2012	32.7%	11.4%	32.7%	17.9%
2013	31.9%	21.8%	41.2%	22.8%
2014	40.0%	19.6%	52.2%	33.0%
2015	50.5%	18.1%	62.9%	45.8%
2016	**48.0%**	**26.3%**	**64.1%**	**50.3%**

Summary - Cross-Origin Communication

4

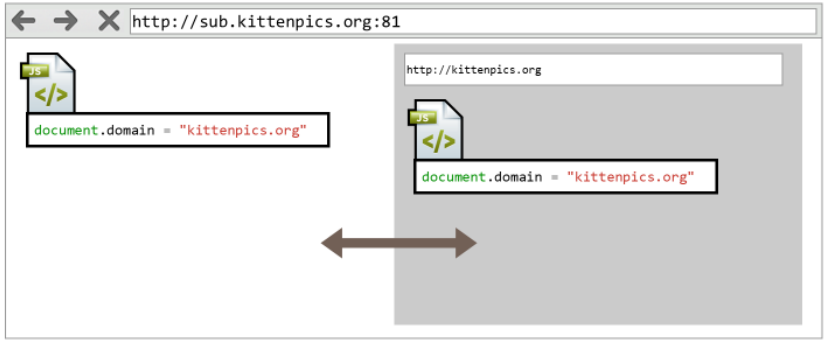
The Same-Origin Policy for JavaScript

- Only allows access if origins match
  - Origin defined by protocol, hostname, and port

Originating document	Accessed document	Non-IE Browser	Internet Explorer
http://example.org/a	http://example.org/b	✓	✓
http://example.org	http://www.example.org	✗	✗
http://example.org	https://example.org	✗	✗
http://example.org	http://example.org:81	✗	✓

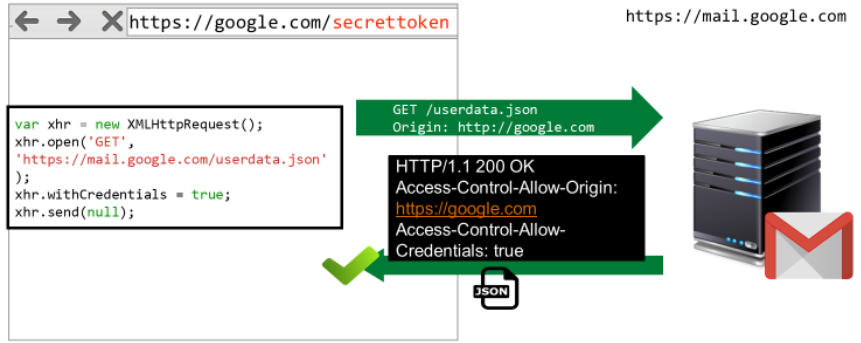
6

Domain Relaxation



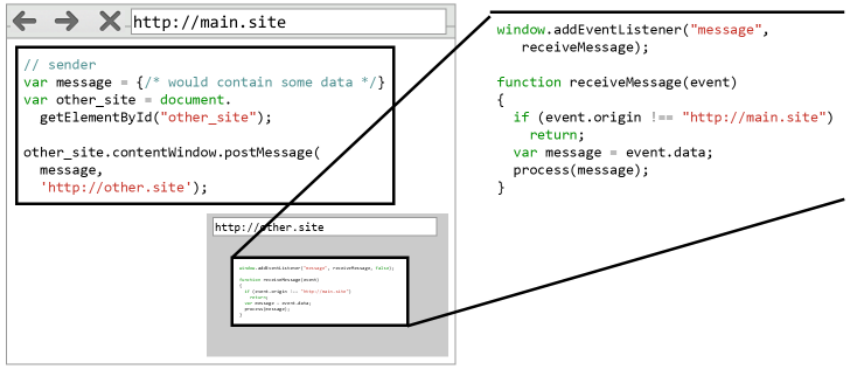
16

CORS Concept (simple request)



27

postMessage Concept



Credits

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