



# **Web Secure Programming**

Lecture 3

Tamara Rezk

---



# Same Origin Policy (high level)

---

Implemented in the browsers :

Full access to same origin resources

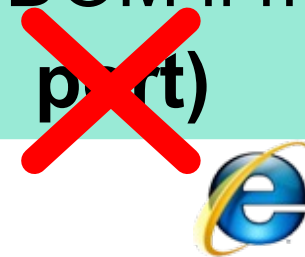
Isolation from different origin resources



# Same origin policy: “high level”

## Same Origin Policy (SOP) for DOM:

- Origin A can access origin B’s DOM if match on  
(**scheme, domain, port**)



## Same Original Policy (SOP) for cookies:

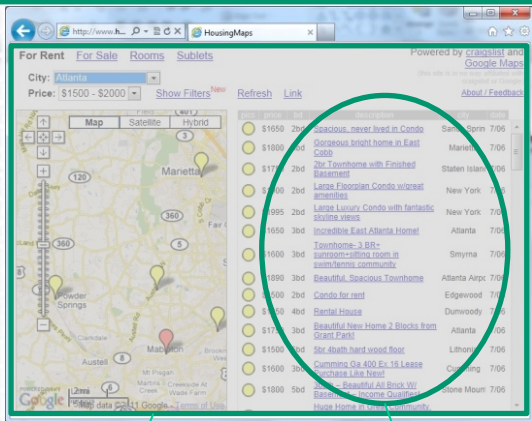
- Generally speaking, based on:  
(**[scheme], domain, path**)

optional

to read: On the Incoherencies in Web Browser Access Control Policies,  
Singh et al, IEEE S&P 2010

# Same Origin Policy for DOM

Using `<script>` tag

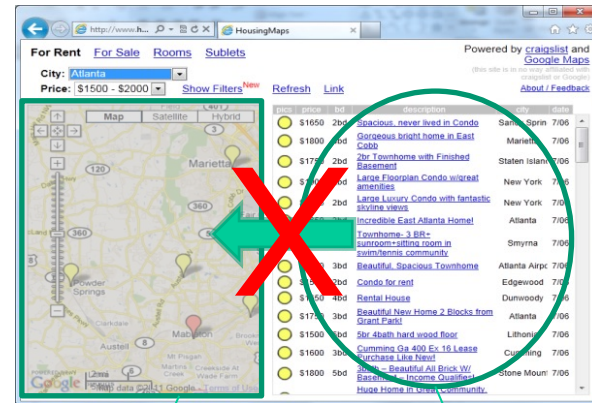


Google Maps Gadget

Integrator's  
Housing Data

- Full sharing (JS Env.)
- Running as integrator
- Gadget trusted

Using `<iframe>` frame



Google Maps Gadget

Integrator's  
Housing Data

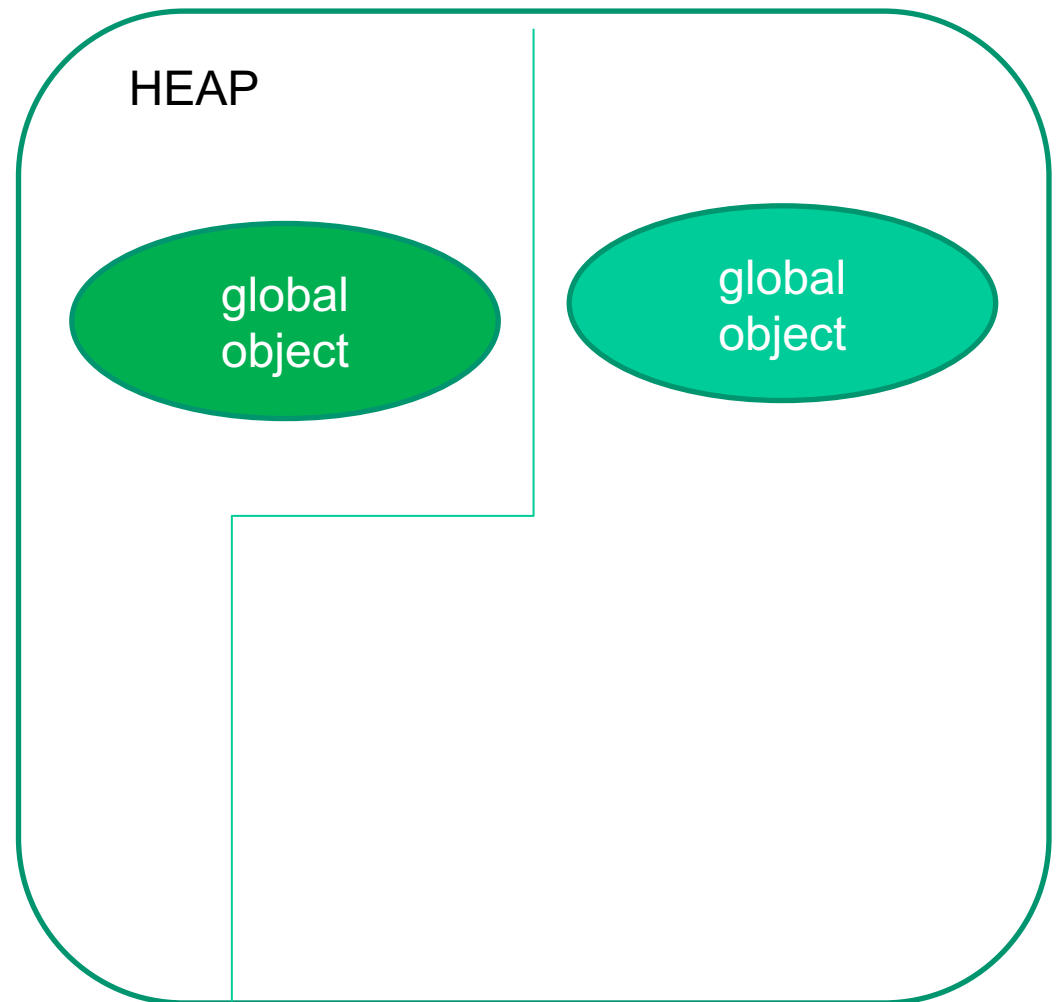
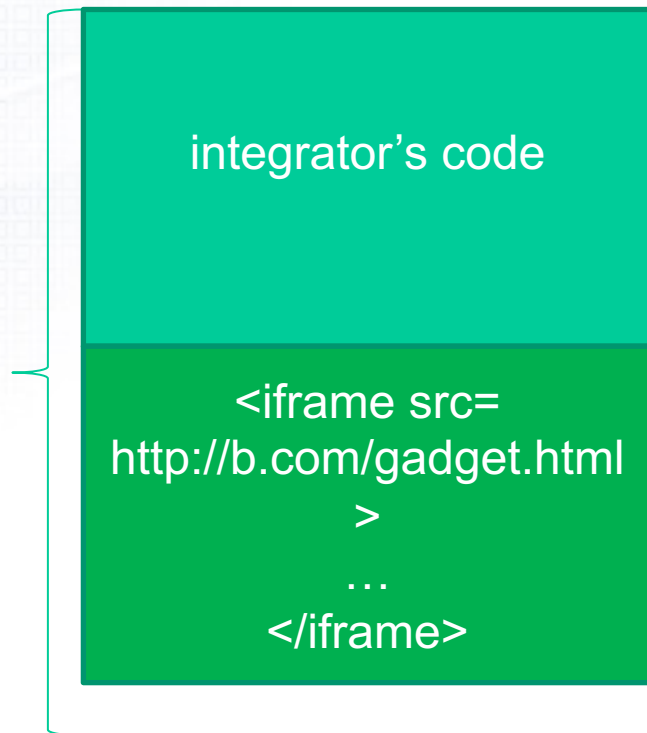
- Full isolation (by SOP)
- Running as gadget
- Limited sharing
  - Frame identifier
  - PostMessage



# The same origin policy (SOP)

- The `<iframe>` tag: Javascript memory

browser







# Frame isolation

- Other origin frames have isolated DOM resources:
- Example (works in Chrome):

`<!-- This is allowed -->`

`<iframe src="SameDomainPage.html"> </iframe>`

`alert(frames[0].contentDocument.body); //works fine`

`<!-- This is **NOT** allowed -->`

`<iframe src="http://google.com"> </iframe>`

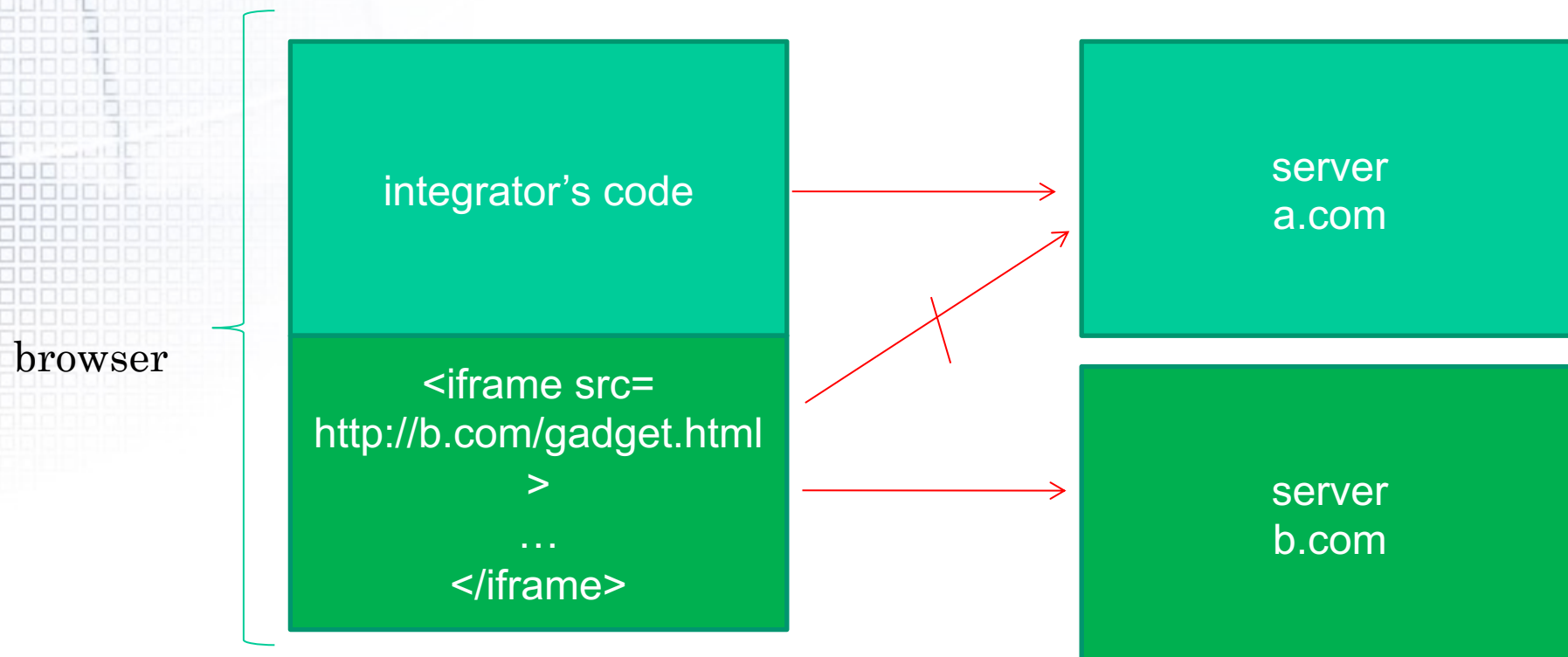
`alert(frames[1].contentDocument.body); //throws error`

Example sop.html



# The same origin policy (SOP)

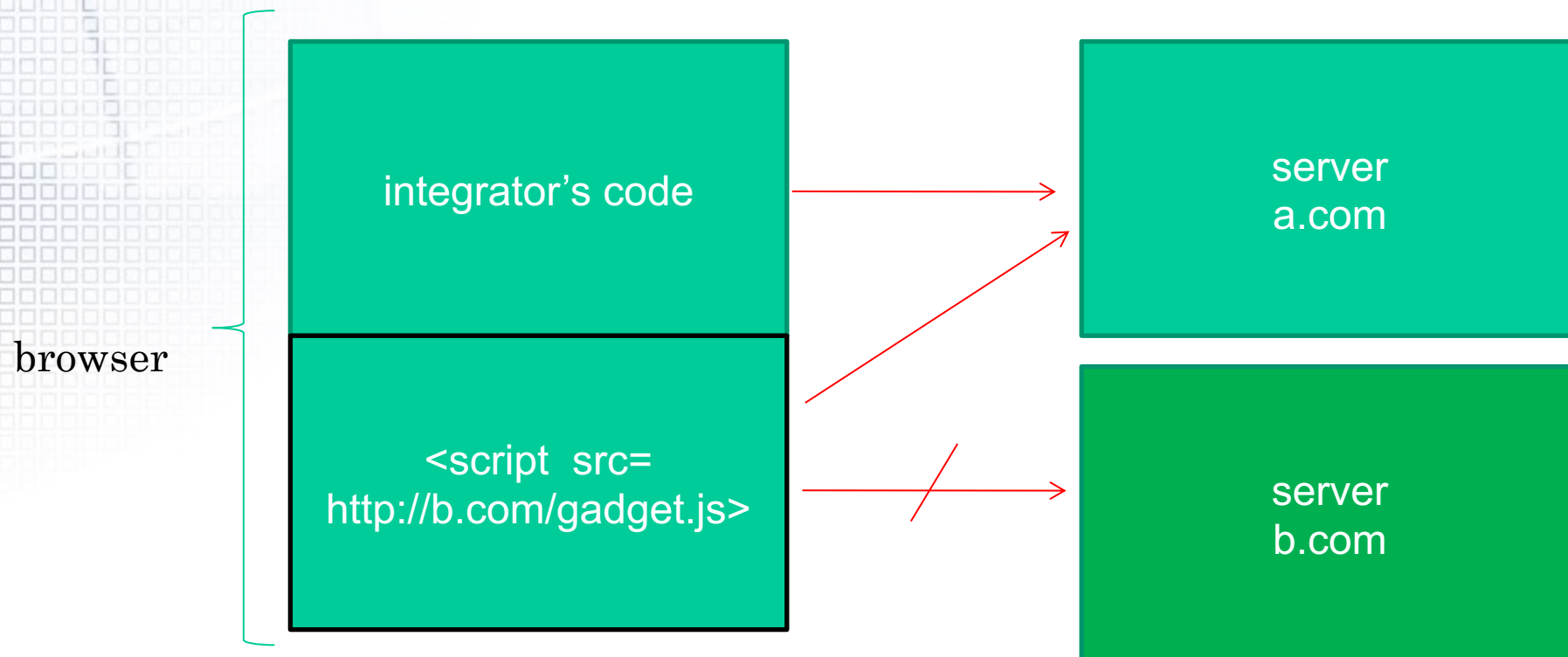
- The `<iframe>` tag: code treated as external code (different origin). The cross domain request is forbidden only before HTML5 (CORS)





# The same origin policy (SOP)

- The `<script>` tag permits to treat code as code from the same origin. The cross domain request is forbidden only before HTML5 (CORS)

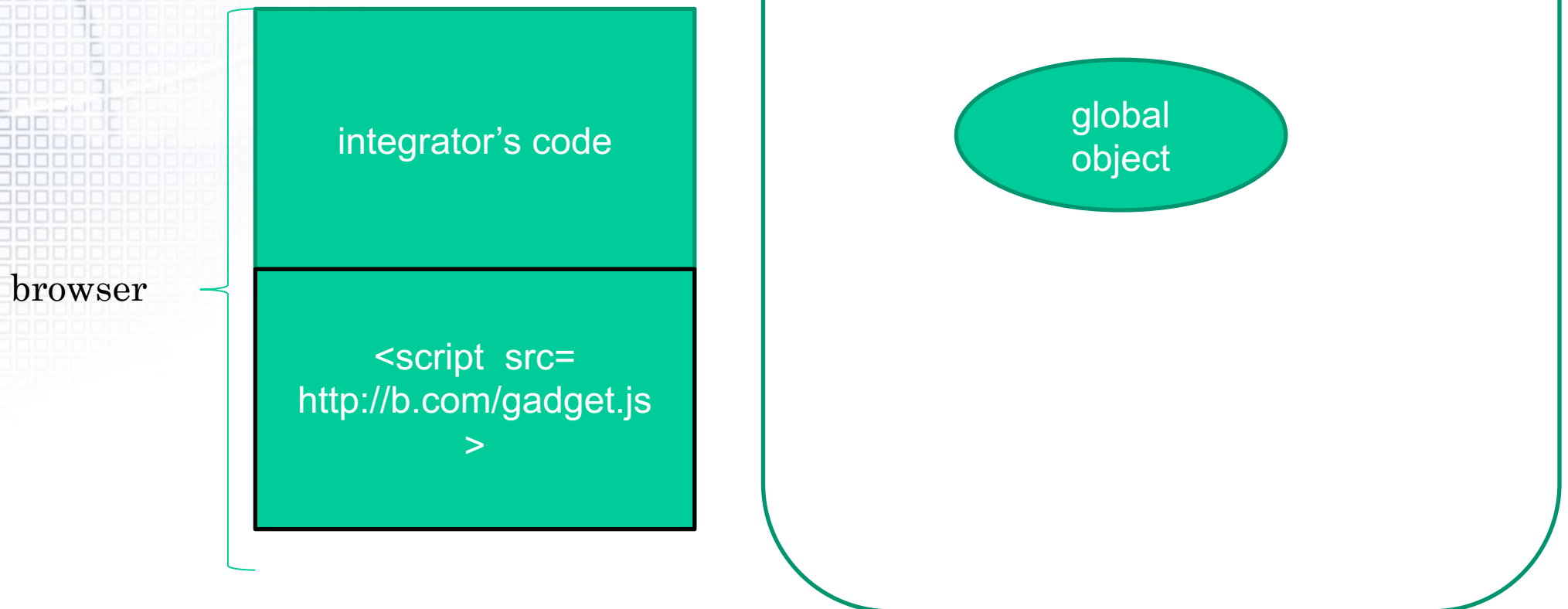






# The same origin policy (SOP)

- The `<script>` tag: what about Javascript behaviour?





# SOP Example

---

Can the iframe see it's parent secret?

```
<div id=secret> The secret is 42 </div>
```

```
<iframe
```

```
src="http://subdomain.host.com/subdomainpage2.ht  
ml">
```

```
</iframe>
```



# SOP Incoherences

---

Can the iframe see its parent secret?

```
<div id=secret> The secret is 42 </div>
```

```
<iframe
```

```
src="http://subdomain.host.com/subdomainpage2.ht  
ml">
```

```
</iframe>
```

And if we change document.domain?



# **Security problems with SOP due to Frame Communication**



# Frame isolation

---

- Other frames cannot access resources from other origins
- Browsers implement a navigation policy that is allowed (changing .location attribute of frame)
  - permissive policy: Guninski attack on CitiBank
  - window policy: gadget hijacking attacks (igoogle+hotmail)



# Frame isolation

---

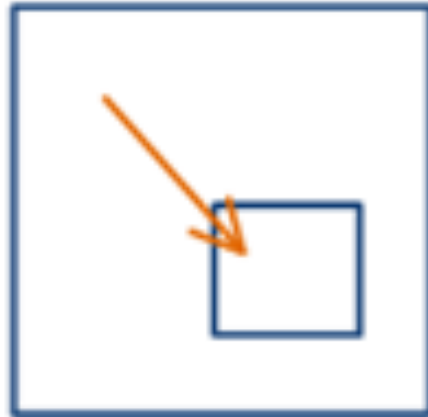
- Other frames cannot access resources from other origins
- Browsers implement a navigation policy that is allowed (changing .location attribute of frame)
  - permissive policy: Guninski attack on CitiBank
  - window policy: gadget hijacking attacks (igoogle+hotmail)
  - descendant policy
  - child policy



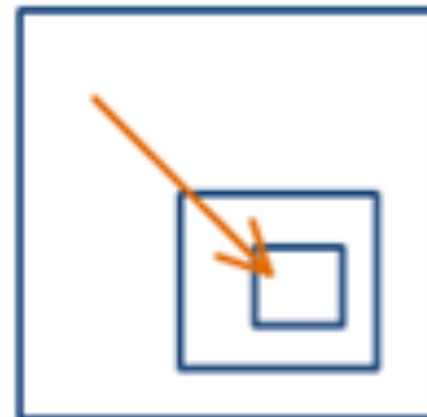


# Navigation policies

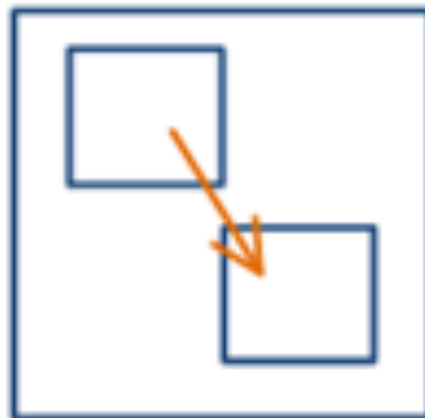
Child Policy



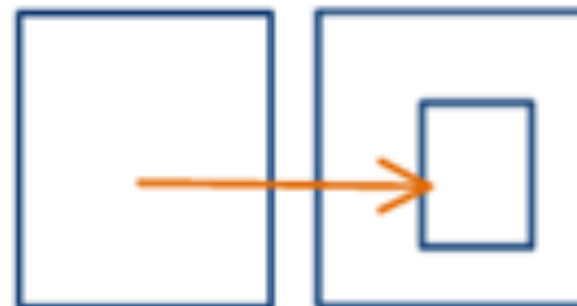
Descendant Policy



Window Policy













Permissive Policy





# Navigation policies

Browser	Policy
 IE 6 (default)	Permissive
 IE 6 (option)	Child
 IE7 (no Flash)	Descendant
 IE7 (with Flash)	Permissive
 Firefox 2	Window
 Safari 2	Permissive
 IE7	Descendant
 Firefox 3	Child
 Safari 3	Child
 HTML 5	Child



# Fragment Identifier Messaging

- Send information by navigating a frame
  - <http://gadget.com/#hello>
- Navigating to fragment doesn't reload frame
  - No network traffic, but frame can read its fragment
- Not a secure channel
  - Confidentiality ✓
  - Integrity ✓
  - Authentication ✗



# HTML 5

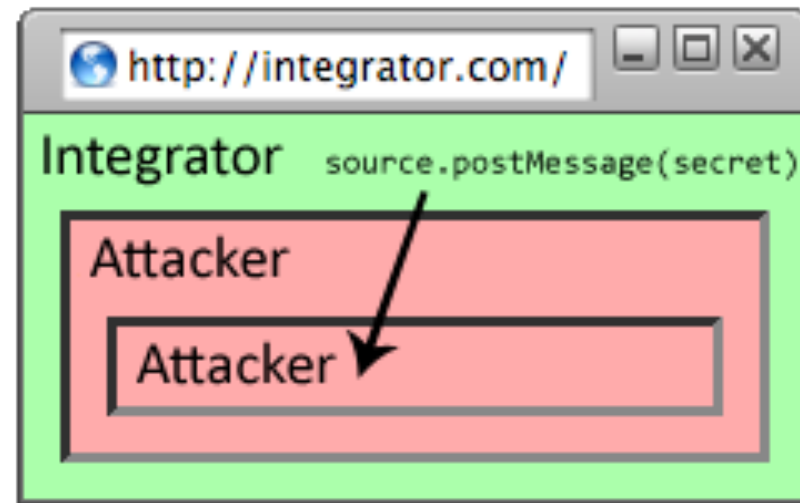
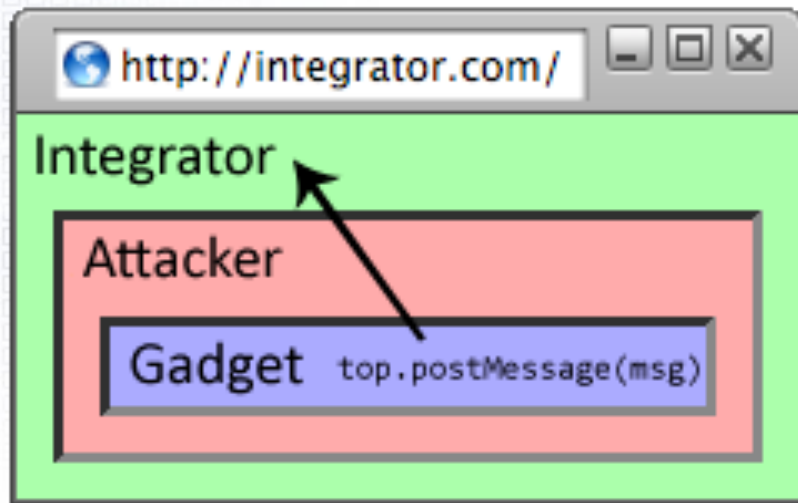
---

- Cross-origin client side communications
- Postmessage channel between frames
- Child policy





# Reply Attack





# Example of PostMessage

```
'http://destination.com'
```

```
window.addEventListener('message', function(event) {  
    if(event.origin !== 'http://originExpected.com')  
    {return;}  
    {console.log('received response: ', event.data);  
    event.source.postMessage('received', event.origin  
});
```

```
frame at 'http://originExpected.com'
```

```
var domain = 'http://destination.com';  
var iframe =  
document.getElementById('myIFrame').contentWindow;  
var message = 'Hello!';  
iframe.postMessage(message, domain);
```





# Security considerations postmessage

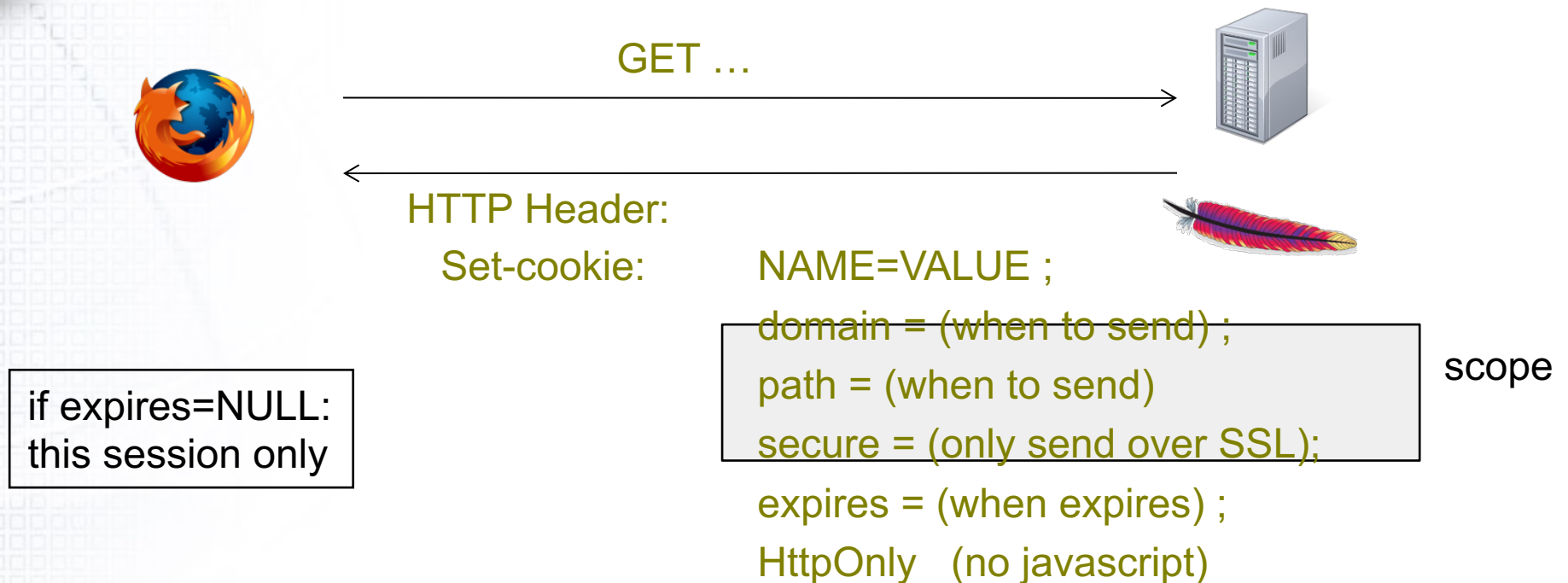
---

- Do not configure target origin (recipient) to “\*”
- Sensitive data can be leaked to unknown widgets
- Always check for sender’s origin
- Always validate data before use
- Do not consume data directly with `eval()` or `innerHTML`



# **SAME ORIGIN POLICY FOR COOKIES**

# Setting/deleting cookies by server



- Delete cookie by setting “expires” to date in past
- Default scope is domain and path of setting URL



# SameSite attribute

Attribute	When cookie fires	Default mode
SameSite=Strict	Domain in URL bar equals the cookie's domain (first-party) AND the link isn't coming from a third-party	n/a
SameSite=Lax	Domain in URL bar equals the cookie's domain (first-party)	New default if SameSite is not set
'SameSite=None'	No domain limitations and third-party cookies can fire	Previous default; now needs to specify 'None; Secure' for Chrome 80



# Who can set a cookie?

domain: any domain-suffix of URL-hostname,  
except TLD

example: host = "login.site.com"

allowed domains

**login.site.com**  
**.site.com**

disallowed domains

**user.site.com**  
**othersite.com**  
**.com**

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

path: can be set to anything



# Cookies are identified by (name,domain,path)

## cookie 1

name = **userid**

value = **test**

domain = **login.site.com**

path = **/**

secure

## cookie 2

name = **userid**

value = **test123**

domain = **.site.com**

path = **/**

secure

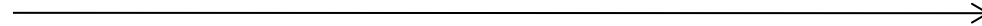
distinct cookies

- Both cookies stored in browser's cookie jar;  
both are in scope of **login.site.com**





# Who can read a cookie?



GET //URL-domain/URL-path  
Cookie: NAME = VALUE



Browser sends all cookies in URL scope:

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



# Examples

both set by **login.site.com**

## cookie 1

name = **userid**

value = **u1**

domain = **login.site.com**

path = **/**

secure

## cookie 2

name = **userid**

value = **u2**

domain = **.site.com**

path = **/**

non-secure

If I type the following URL in my browser,  
which cookie will the browser send?

<http://checkout.site.com/>

**cookie: userid=u2**

<http://login.site.com/>

**cookie: userid=u2**

<https://login.site.com/>

**cookie: userid=u1; userid=u2**

(arbitrary order)



# Client side read/write: **document.cookie**

Let's see cookies in browser!

- Setting a cookie in Javascript:  
`document.cookie = "name=value; expires=...; "`
- Reading a cookie: `alert(document.cookie)`  
prints string containing all cookies available for document (based on [protocol], domain, path)
- Deleting a cookie:  
`document.cookie = "name=; expires= Thu, 01-Jan-70"`

Example sop3.php



# **When a server sees a cookies it**

- Does not see cookie attributes (e.g. secure)
- Does not see which domain set the cookie

Server only sees:      Cookie: NAME=VALUE

## **This server blindness carries some problems**



# Example : server does not see which domain set the cookie

- Alice logs in at **login.site.com**  
login.site.com sets session-id cookie for **.site.com**
- Alice visits **evil.site.com**  
overwrites .site.com session-id cookie with session-id of user “badguy”
- Alice visits **homework.site.com** to submit homework.  
homework.site.com thinks it is talking to “badguy”





# Interaction with the DOM SOP

Cookie SOP: path separation

**x.com/A** does not see cookies of **x.com/B**

Not a security measure:

DOM SOP: **x.com/A** has access to DOM of **x.com/B**

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

Path separation is done for efficiency not security:

**x.com/A** is only sent the cookies it needs





# TP

---

- 1) Look at the code of integrator.html and write code for evilGadget.js in such a way that evilGadget.js will send the secret to evil.com (simulate evil.com with another port). Rewrite integrator.html so the same origin policy will protect the secret.
- 2) Look at the sop2.html. From subdomain2.html try to read the secret from the integrator, what happens according to SOP? How do you read the secret by using document.domain?



# TP cont.

---

3. Write two different services from the same server that set a cookie. On the client side include a gadget and try the following things:
- let the gadget delete the cookie via JavaScript
  - Can the second service delete the cookie of the first? Justify why.
  - let the gadget send the cookie to another server (you can use a different port to simulate this)
  - Does the previous item work if the gadget is inside a frame?
  - and if the gadget is inside a script and the cookie is initially set as httpOnly?
  - and if the gadget is inside a script and the cookie is initially set as secure?

Justify all your answers with code and explanations.