Sandboxing JavaScript

Nick Nikiforakis & Steven Van Acker

SECAPPDEV 2013






KU LEUVEN
DISTRINET RESEARCH GROUP
iMinds
CONNECT.INNOVATE.CREATE

Overview

- **Part 1:** State-of-practice usage of 3rd party JavaScript
  - Nick Nikiforakis
- **Part 2:** Variety of policies and enforcement techniques for sandboxing JavaScript
  - Steven Van Acker

Sandboxing JavaScript

Part 1: State-of-practice usage of 3rd party JavaScript

SECAPPDEV 2013






KU LEUVEN
DISTRINET RESEARCH GROUP
iMinds
CONNECT.INNOVATE.CREATE

Who am I ?

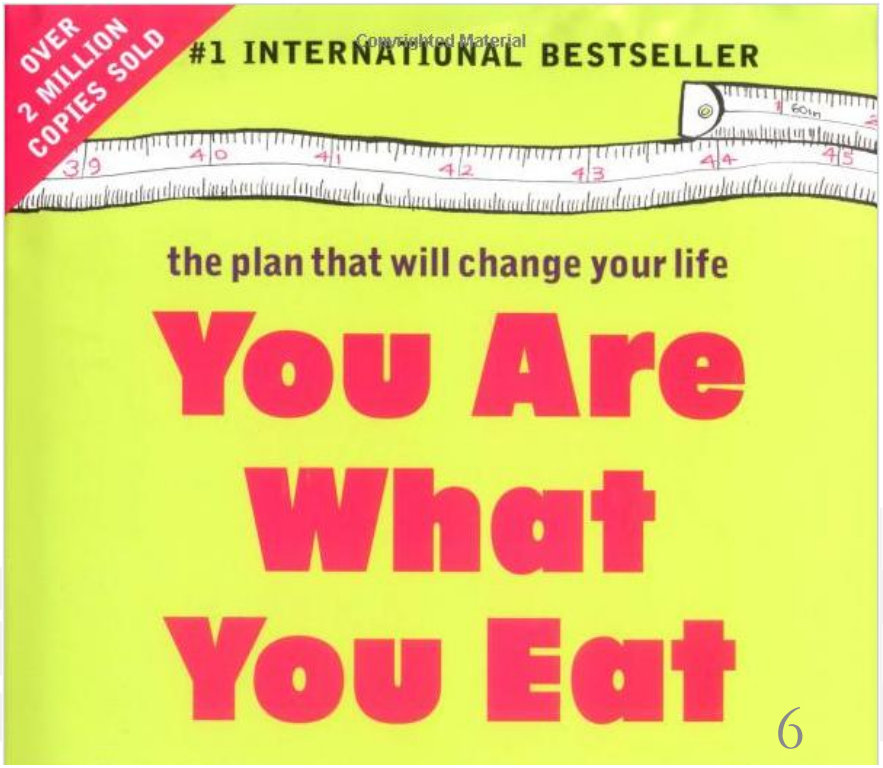
- Nick Nikiforakis
  - nick.nikiforakis@cs.kuleuven.be
  - http://www.securitee.org
- PhD researcher @ KU Leuven
- Experience with the analysis of large-scale online ecosystems, from a security and privacy perspective
- Bypassed:
  - Chrome's Anti-XSS mechanism
  - McAfee's Social Protection

Who needs sandboxing anyway?

- If you only trust scripts from well-known vendors, you don't have anything to worry about… right?
- Wrong!
  - There is no unhackable vendor
  - Scripts can include other scripts… from wherever
  - …

You are what you include…

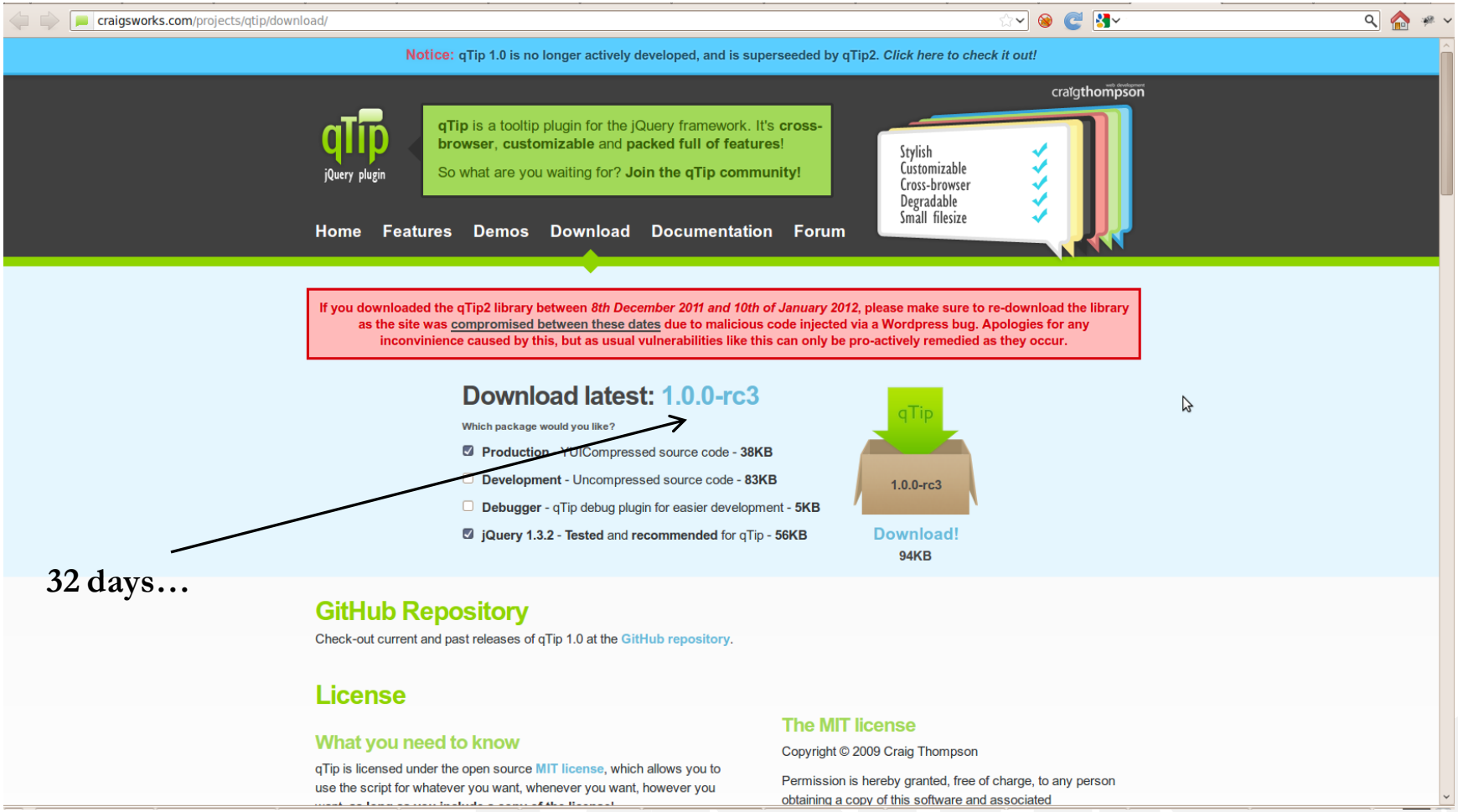
Nikiforakis et al. **"You Are What You Include: Large-scale evaluation of remote JavaScript Inclusions"**, CCS 2012



Remote JS-Providing hosts

- Given that sites include remote JS, which third-party vendors do they currently trust?
- What is the maintenance profile of each JS provider?
  - → Could a provider be attacked as a way of reaching a harder-to-get target?
- Are there attack vectors, in relation to remote inclusions, that we were not aware of ?
- How can one protect his web application?
  - → Are coarse-grained sandboxes sufficient?

Motivation…



32 days…

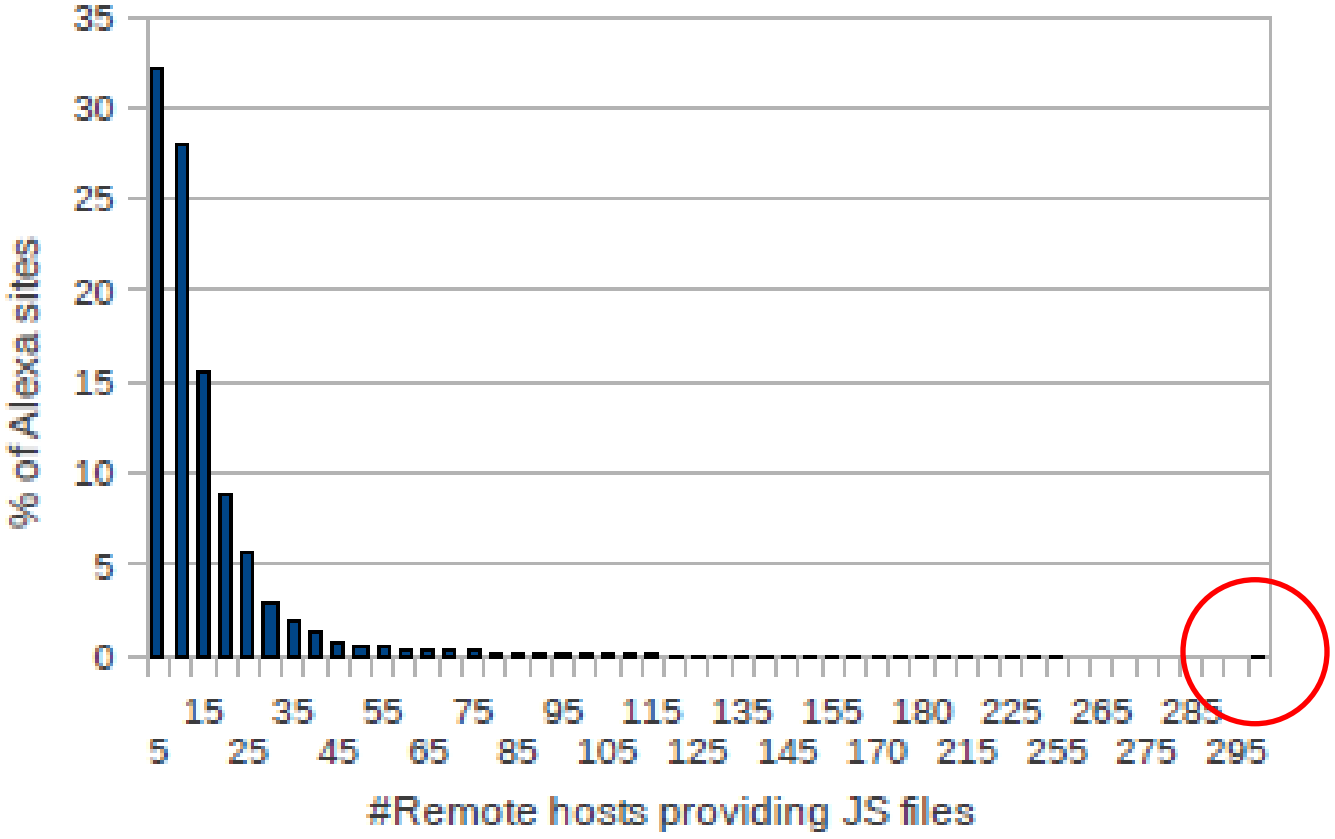
Data Collection

- Discovering remote JavaScript inclusions (aka trust relationships)
- Alexa Top 10,000
  - → Up to 500 pages from each
  - → Pages chosen by Bing
    - Query “site:google.com”

Crawling results

- Crawled over 3,300,000 pages belonging to the Alexa top 10,000
- Discovered:
  - 8,439,799 remote inclusions
  - 301,968 unique JS files
  - 20,225 uniquely-addressed remote hosts
    - Addressed by domain-name
    - Addressed directly by IP address

How many remote hosts?



Remote IP inclusions

- 0.27% of the inclusions found were addressing a remote host by its IP address
  - → 299 Alexa domains addressing 324 unique IP addresses
- Most of them in China (35.18%)
- Only 65 unique cases of cross-country IP-based inclusions

Popular JavaScript libraries

Offered service	JavaScript file	% Top Alexa
Web analytics	www.google-analytics.com/ga.js	68.37%
Dynamic Ads	pagead2.googlesyndication.com/pagead/show_ads.js	23.87%
Web analytics	www.google-analytics.com/urchin.js	17.32%
Social Networking	connect.facebook.net/en_us/all.js	16.82%
Social Networking	platform.twitter.com/widgets.js	13.87%
Social Networking & Web analytics	s7.addthis.com/js/250/addthis_widget.js	12.68%
Web analytics & Tracking	edge.quantserve.com/quant.js	11.98%
Market Research	b.scorecardresearch.com/beacon.js	10.45%
Google Helper Functions	www.google.com/jsapi	10.14%
Web analytics	ssl.google-analytics.com/ga.js	10.12%

Popular JavaScript libraries

Offered service	JavaScript file	% Top Alexa
Web analytics	www.google-analytics.com/ga.js	68.37%
Dynamic Ads	pagead2.googlesyndication.com/pagead/show_ads.js	23.87%
Web analytics	www.google-analytics.com/urchin.js	17.32%

MINDED SECURITY BLOG

MINDED SECURITY RESEARCH LABS

TUESDAY, SEPTEMBER 25, 2012

ABOUT MINDED SECURITY

Temporary Patch for a Dom Xss 0day in Addthis.com Widgets

Note: Addthis.com fixed this issue yesterday 26/09/12 thanks to Addthis.com team for fixing it so rapidly!

Minded security

Security analysis

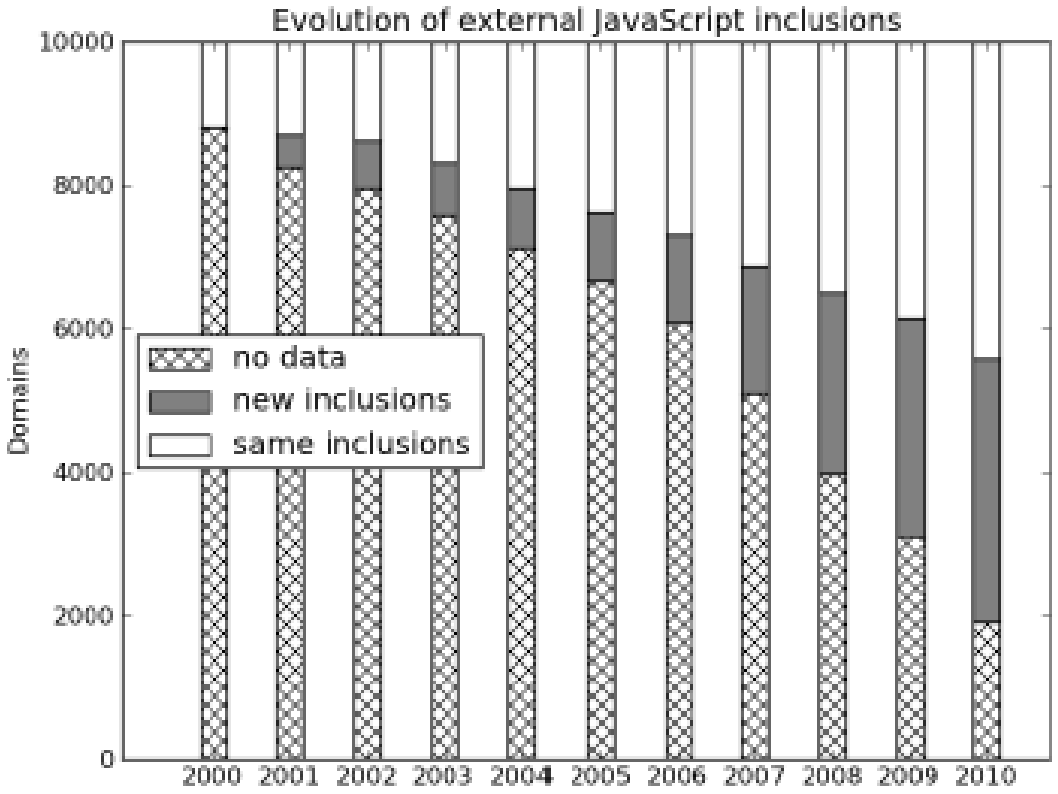
- Are sites trusting more, or less remote hosts as time goes by?
  - → Evolution of external JavaScript inclusions
- Are the ones who are currently trusting, worthy of their trust?

Evolution of inclusions

Using *archive.org*, crawl the page that is:

a) Available throughout the years

b) Has the most inclusions in our current dataset



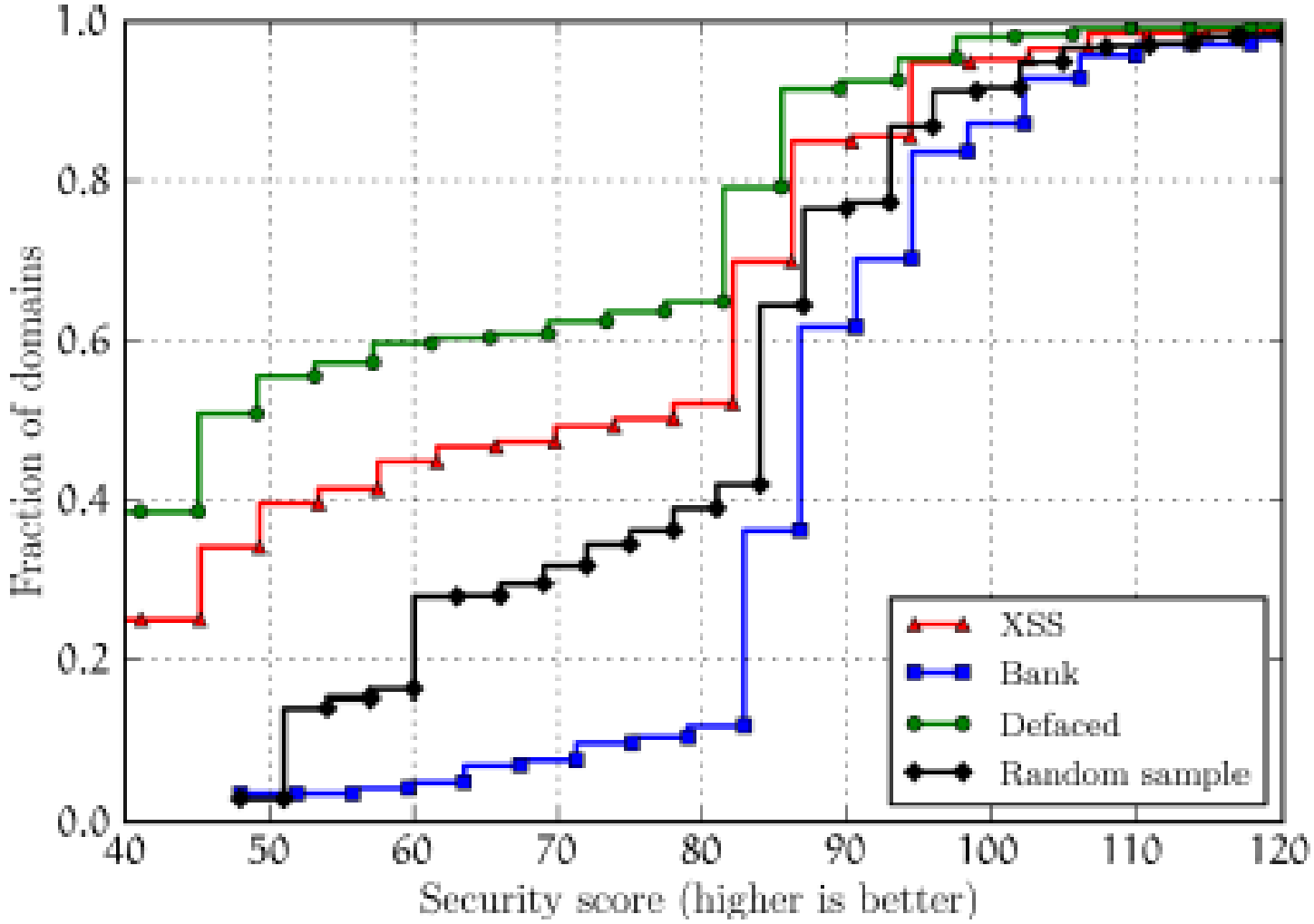
Designing a quality-of-maintenance metric

- Assumption: Unkempt third-party providers are easier to attack
  - Availability: DNS not expired, publicly-routable IP address
  - Cookies (at least one):
    - HttpOnly?
    - Secure?
    - Path & Expiration?
  - Anti-XSS & Anti-Clickjacking headers?
  - Cache control
  - SSL implementation
    - Weak ciphers
    - Valid certificates
    - Strict Transport Protocol
  - Outdated web servers?

Weights and Training

- Supervised learning needs a training-set, a ground truth
  - We had none
- Common logic:
  - If you expect site A to be more secure than site B, then the metric should reflect that
  - Data-sets:
    - XSSed
    - Defaced
    - Banks
    - Random sites

Results


1.0
0.8
0.6
0.4
0.2
0.0
Fraction of domains
40
50
60
70
80
90
100
110
120
Security score (higher is better)
XSS
Bank
Defaced
Random sample

Bad apples

- cafemom.com
  - Invalid SSL certificate
  - Non-httponly & non-secure cookies
  - Both HTTP & HTTPS work
- criteo.com (included by 117)
  - Weak SSL ciphers
  - weak DH key exchange
- levexis.com (included by 15)
  - Invalid SSL certificate

Attacks?

- In about 8.5 million records of remote inclusions, is there something that we didn't know?
- 4 Things! ☺
  - Cross-user & Cross-network Scripting
  - Stale domain-based inclusions
  - Stale IP-based inclusions
  - Typo-squatting Cross-Site Scripting

Cross-user Scripting

- <script src=http://localhost/script.js>
  - 133 records were found
  - 131 specified a port (localhost:12345), always greater than 1024
  - Attack:
    - Setup a web-server, listen to high ports, hack other users

Cross-network Scripting

- <script src=http://192.168.2.3/script.js>
  - 68 of them
  - Same as before, but now you just need to be in the same local network
- Who is doing that?
  - akamai.com
  - virginmobileusa.com
  - gc.ca (Government of Canada)

Stale domain-based inclusions

- What happens when you trust a remote site and the domain of that site expires?
  - Anyone can register it, and start serving malicious JS
  - Equal in power to the, almost extinct, stored XSS
    - Try proving in court that someone hacked you with that
- 56 domains found, used in 47 sites
  - Some were identified as special cases

Shopping spree!

- Registered some of the stale domains:
  - blogtools.us -> goldprice.org (4,779th in Alexa)
  - hbotapadmin.us -> hbo.com

	Blogtools.us	Hbotapadmin.com
Visits	80,466	4,615
Including domains	24	4
Including pages	84	41

Stale IP-based remote inclusions

- What if the IP address of the host which you trust for JavaScript, changes?
  - The including page's scripts must also change
  - Do they?
- Manual analysis of the 299 pages
  - 39 addresses had:
    a) Not changed
    b) no longer provided JavaScript
       a) In 89.74%, we got a "Connection Timeout"

Typosquatting XSS (TXSS)

- Typosquatting
  - registering domains that are mistypes of popular domains
  - Serve ads, phishing, drive-by downloads etc. to users that mistype the domain
- Unfortunately… developers are also humans
  - <script src=http://googlesyndicatio.com/…>

Examples found…

Intended domain	Actual domain
googlesyndication.com	googlesyndicatio.com
purdue.edu	purude.edu
worldofwarcraft.com	worldofwaircraft.com
lesechos.fr	lessechos.fr
onegrp.com	onegrp.nl

	Googlesyndicatio.com
Unique visitors	163,188
Including domains	1,185
Including pages	21,830

Countermeasures

- Problems with remote inclusions
  - A developer can mess up
    - Cross-user, cross-network and TXSS
  - The remote host can mess up
    - Low security, expiration of domain names
- How to protect one's self?
  i. Sandbox remote scripts
  ii. Download them locally

Coarse-grained sandboxing

- Is it feasible?
- What are the current requirements of legitimate scripts?
- Study the top 100
  - → Automatically study each script
    - JavaScript wrappers + stack trace
  - → Find out what sensitive resources they access
    - Cookies, Storage, Geolocation, Eval, document.write
  - → Is coarse-grained containment possible?

Access to resources

JS Action	# of Top scripts
Reading Cookies	41
`document.write()`	36
Writing Cookies	30
`eval()`	28
`XHR`	14
Accessing LocalStorage	3
Accessing SessionStorage	0
Geolocation	0

Using local copies

- Study the frequency of script modifications
  - Discover overhead for administrator
- Top 1,000 most-included scripts (803)
  - Download every script three consecutive times and remove the ones that changed all three times
  - Study the rest for a week
- 10.21% were modified
  - 6.97% were modified once
  - 1.86% were modified twice
  - 1.83% were modified three or more

96.76% were modified at most once

More treats…

- Things not mentioned:
  - Mistakes with single (/) and double slashes (//)
    - //foo.js VS /foo.js
  - Mistypes that are already registered and could be poisoning the including sites
- Some registrars reclaim the domains that were used in attacks and, after a while, put them back into circulation

Conclusions

- Remote inclusions mean, almost unconditional, trust
  - → Some trust TOO much
- Things can go wrong:
  - → Because of the includer
  - → Because of the includee
- 4 new attack vectors
- "Easy" solutions, like coarse-grained sandboxing, may not be as effective as hoped for

End of Part 1

Over to
Steven!

Sandboxing JavaScript

Part 2: Variety of policies and enforcement techniques for sandboxing JavaScript

SECAPPDEV 2013






KU LEUVEN
DISTRINET RESEARCH GROUP
iMinds
CONNECT.INNOVATE.CREATE

Who am I ?

- Steven Van Acker
  - Steven.VanAcker@cs.kuleuven.be
  - Twitter: @StevenVanAcker
- PhD researcher @ KU Leuven
- Experience with:
  - large-scale internet research (FlashOver, You are what you include)
  - JavaScript sandboxing (WebJail, JSand)
  - Building/maintaining/attacking secure systems (OverTheWire.org, CTF)

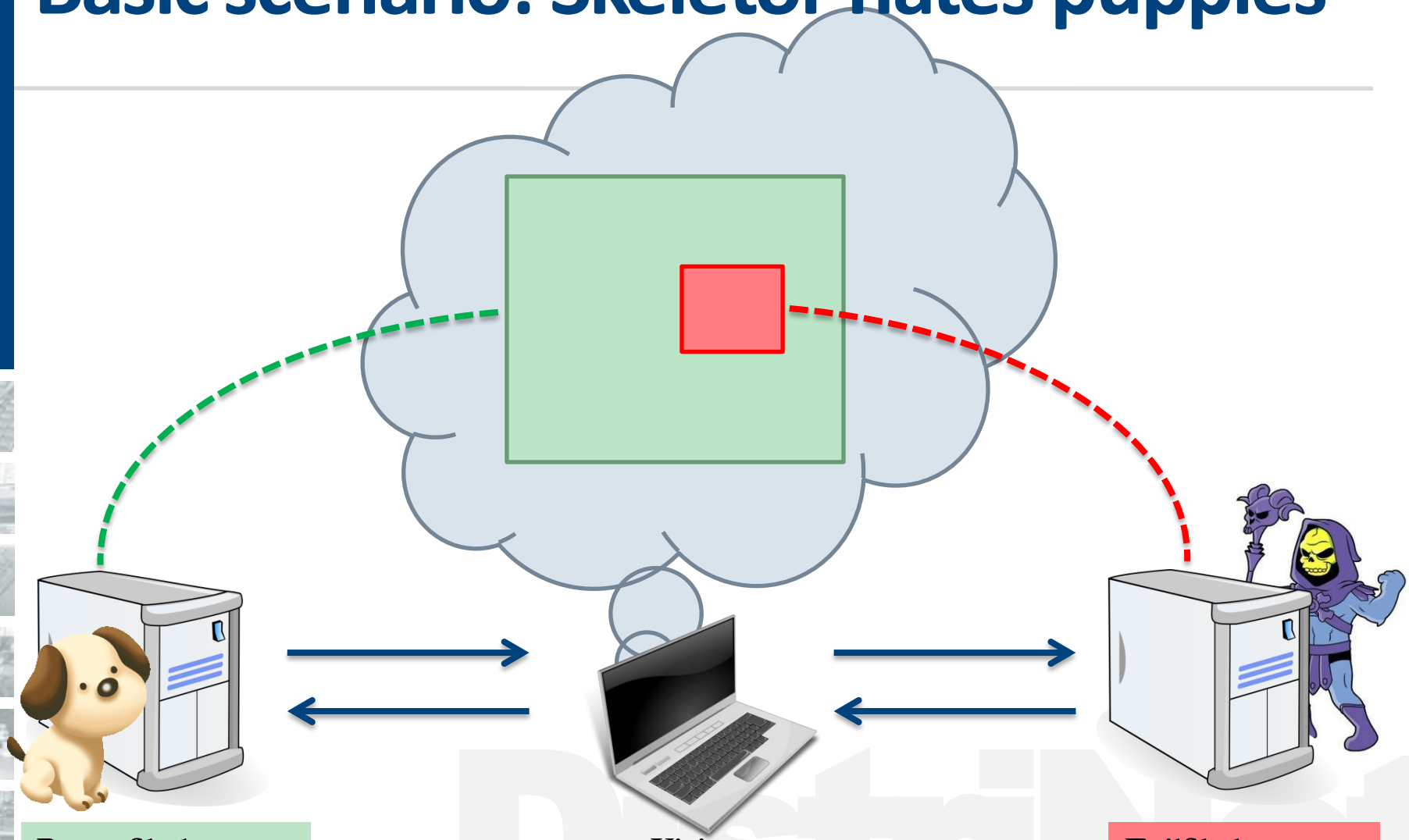
A view of the world

Example scenarios and policies

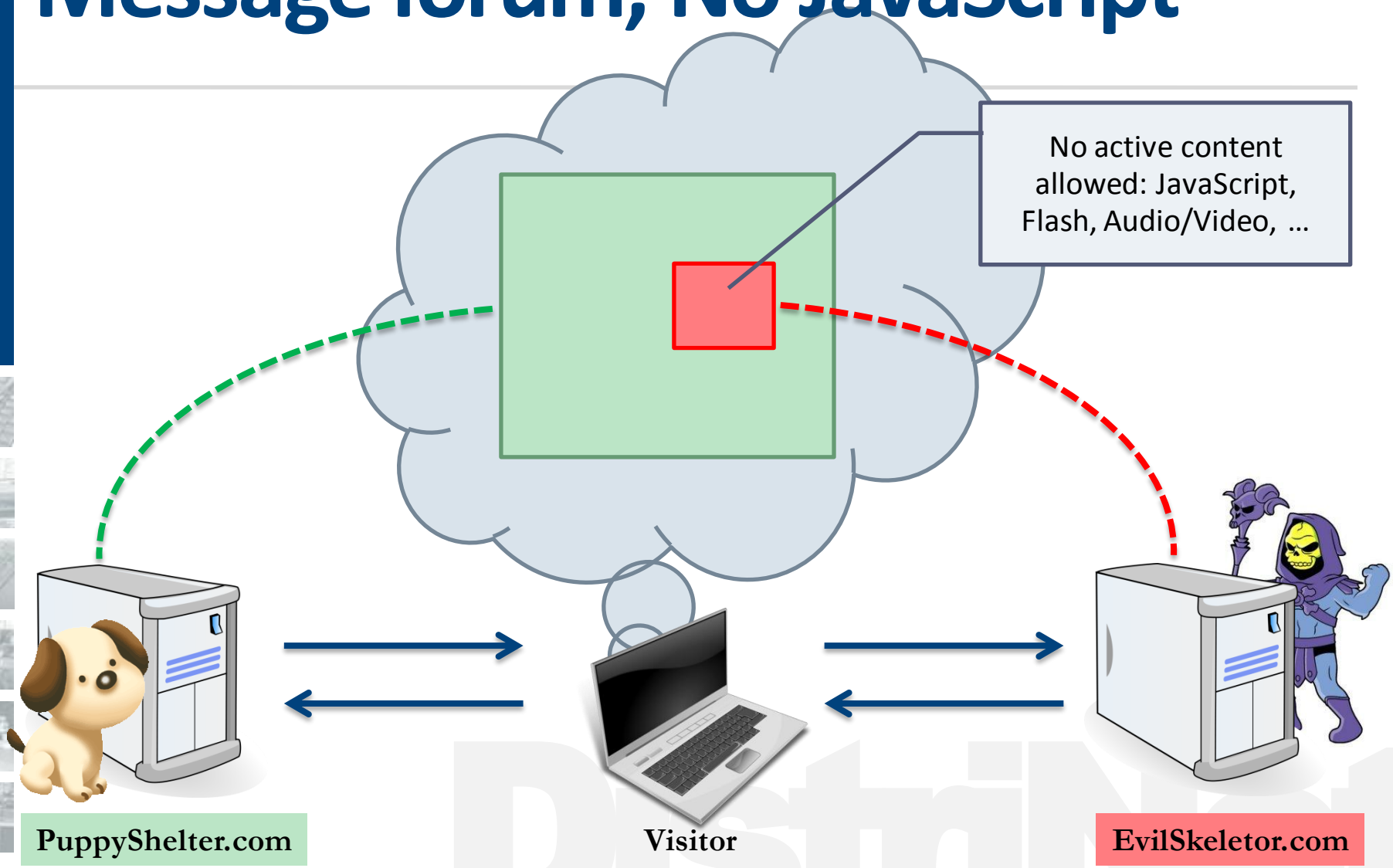




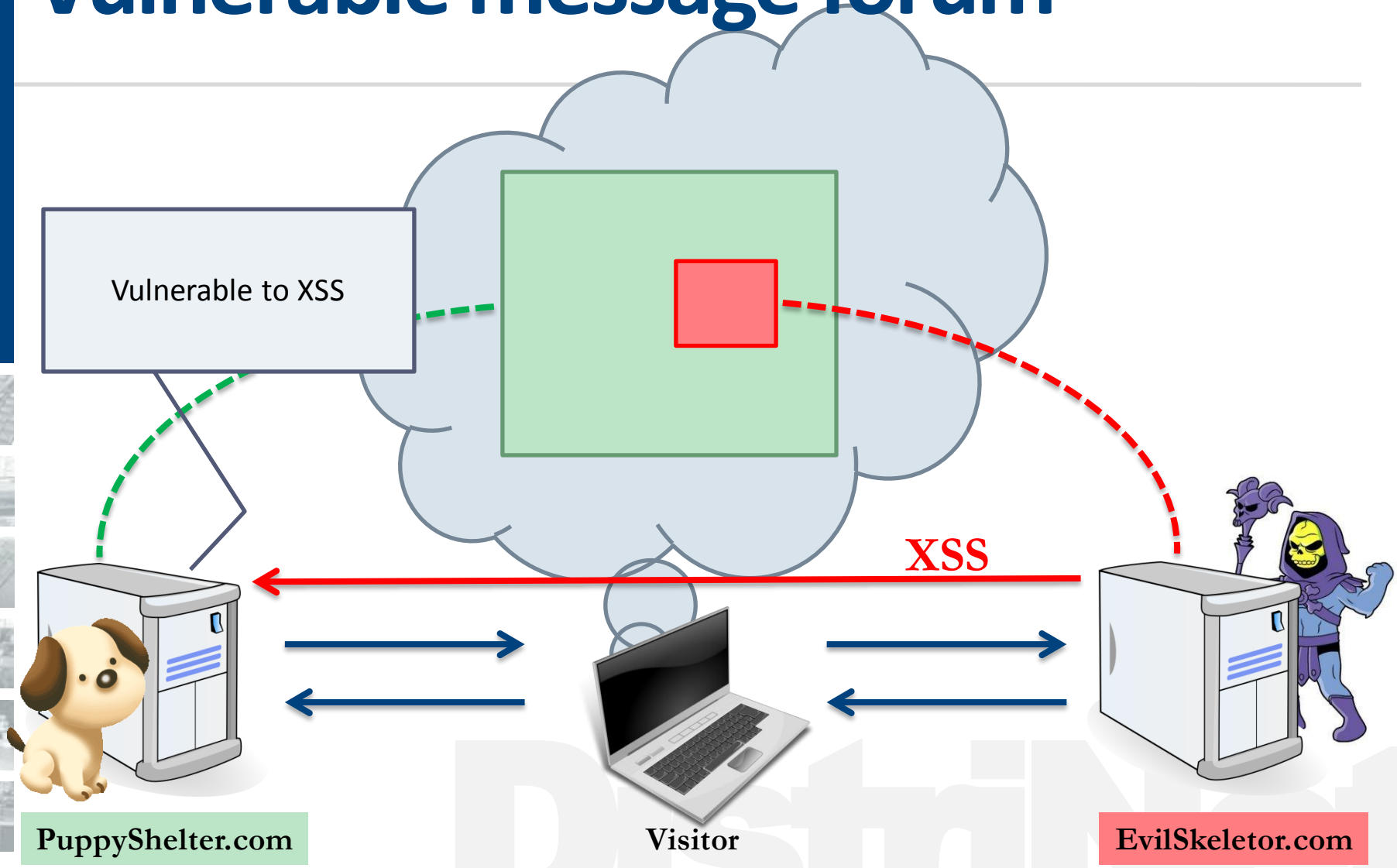

Basic scenario: Skeletor hates puppies



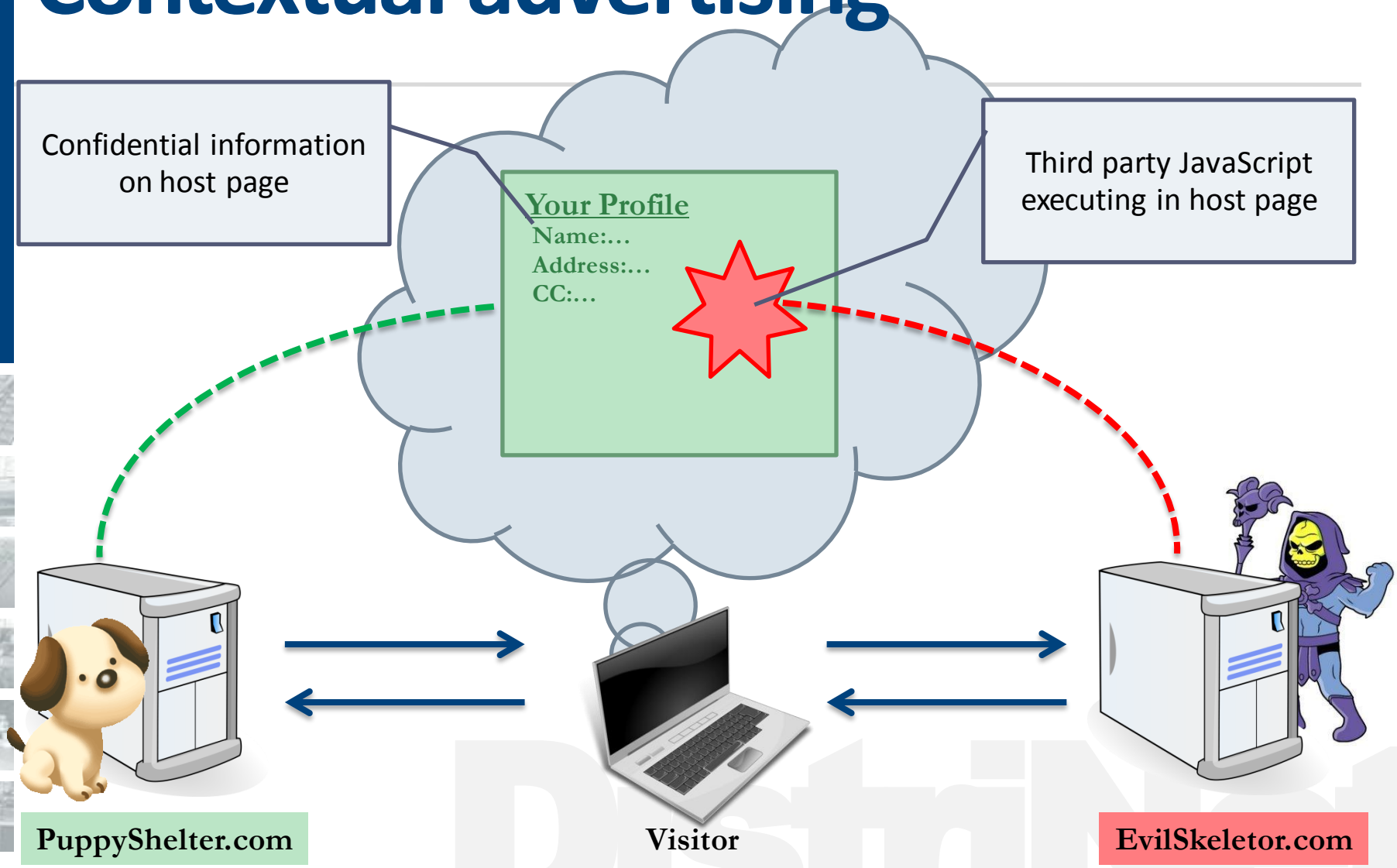
Message forum, No JavaScript



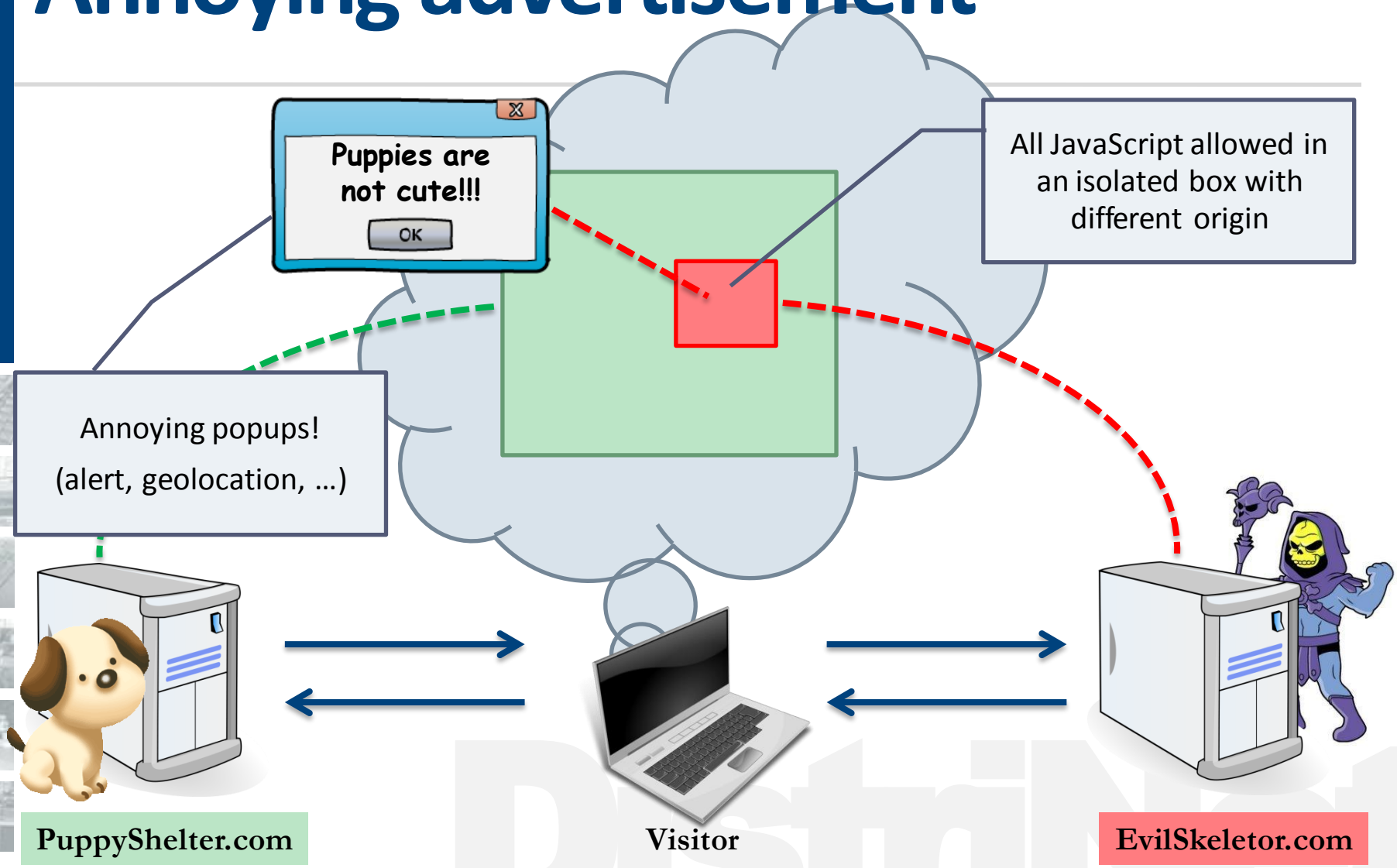
Vulnerable message forum



Contextual advertising



Annoying advertisement

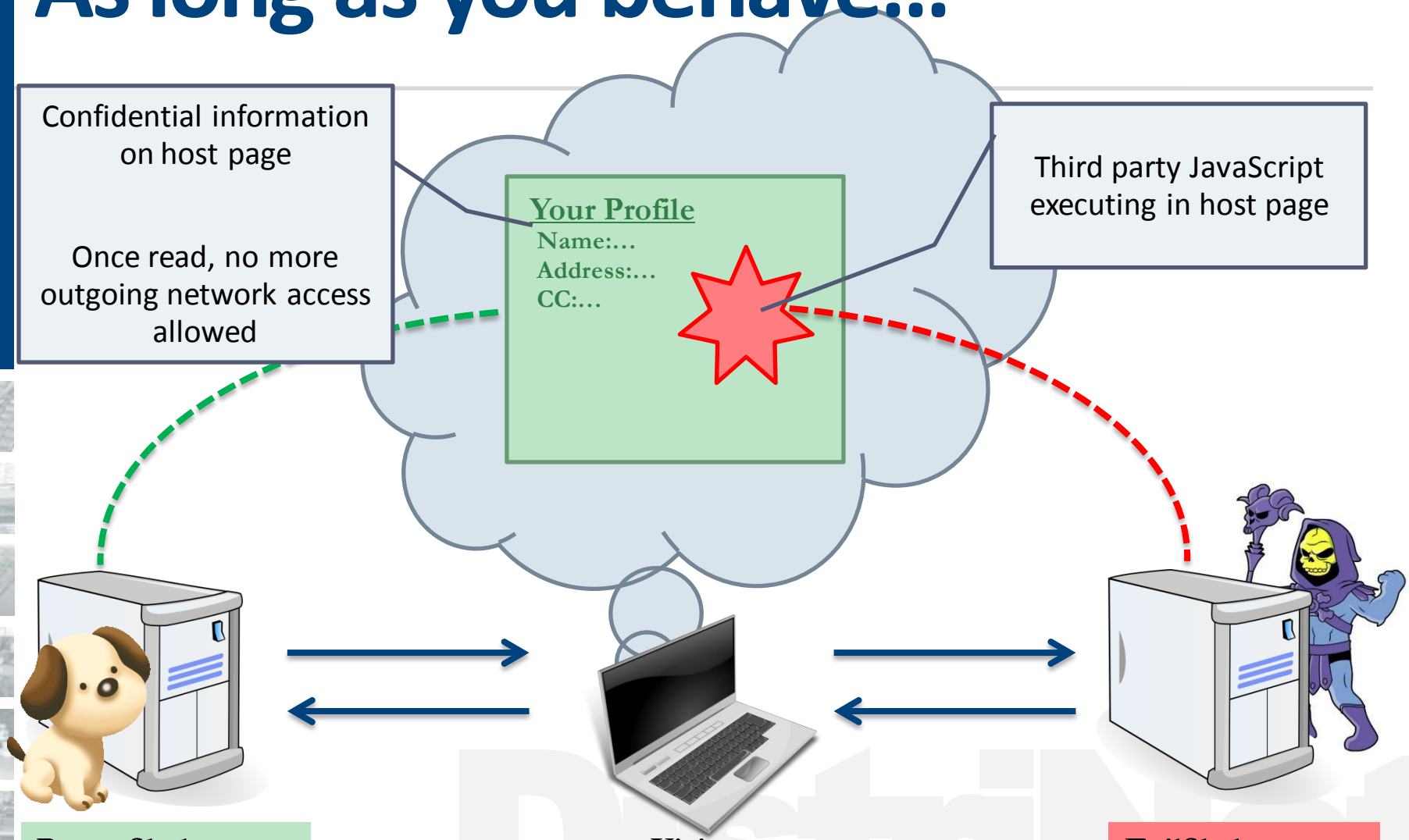


As long as you behave…

Confidential information on host page

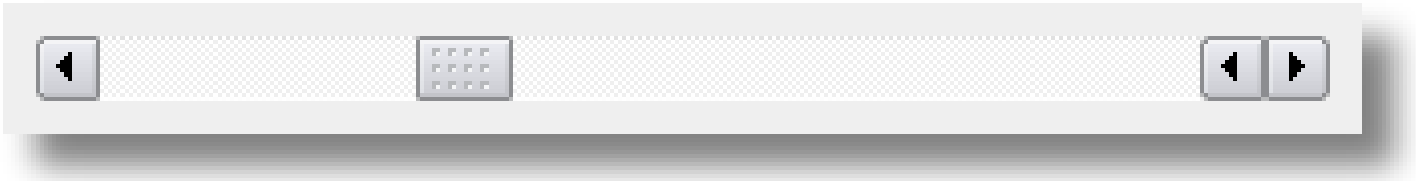
Once read, no more outgoing network access allowed

Third party JavaScript executing in host page



Types of policies

- Policy granularity
  - From very simple and coarse (JS on/off)
  - To very complex and fine-grained (disable network access after cookie-read)



Simple
Coarse

Complex
Fine-grained

Control all the things!

What types of access can we control?

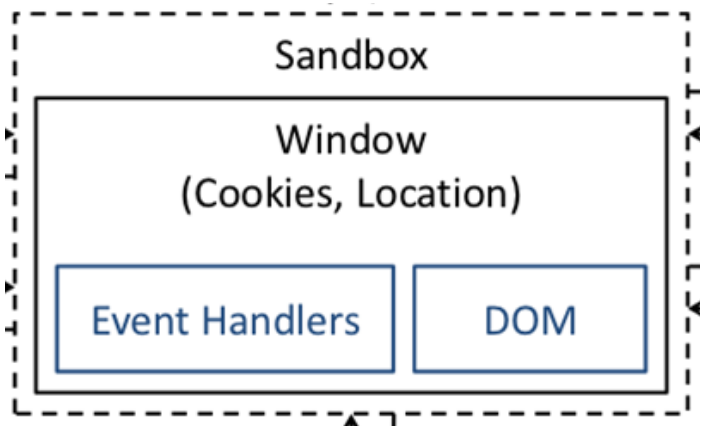




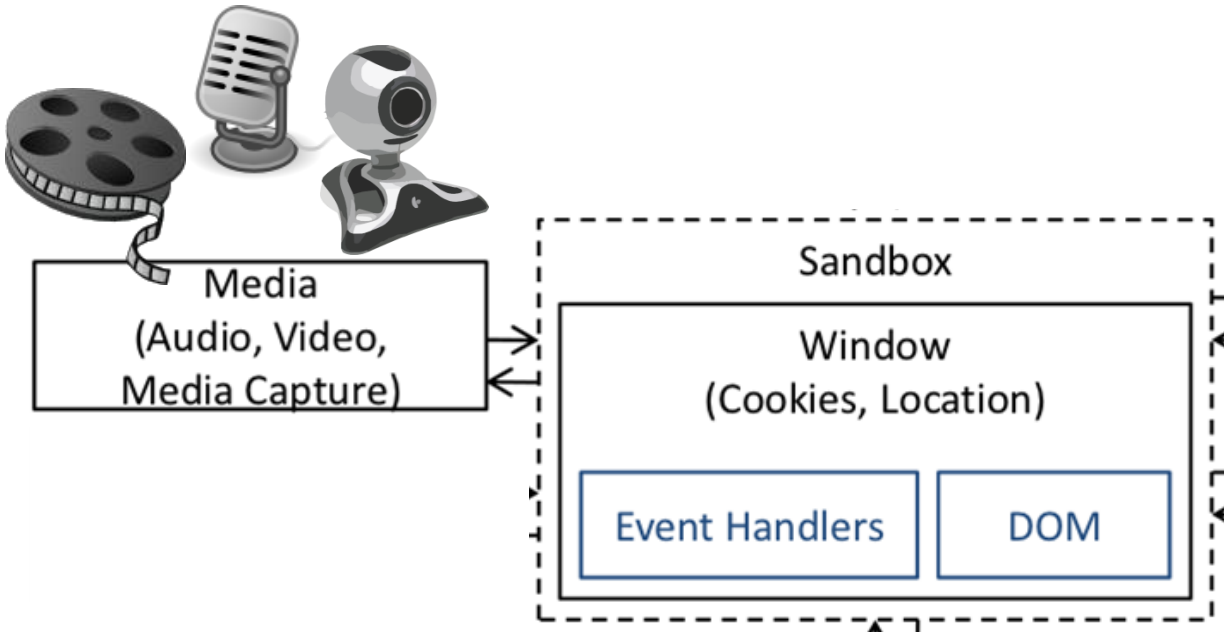

Same-origin policy vs. functionality

- By default, most access is limited by the same origin policy
  - <scheme>://<host>:<port>
  - Examples: XHR, DOM Access, cookies, …
- Some functionality is not bound by SOP
  - Geolocation, alert(), …

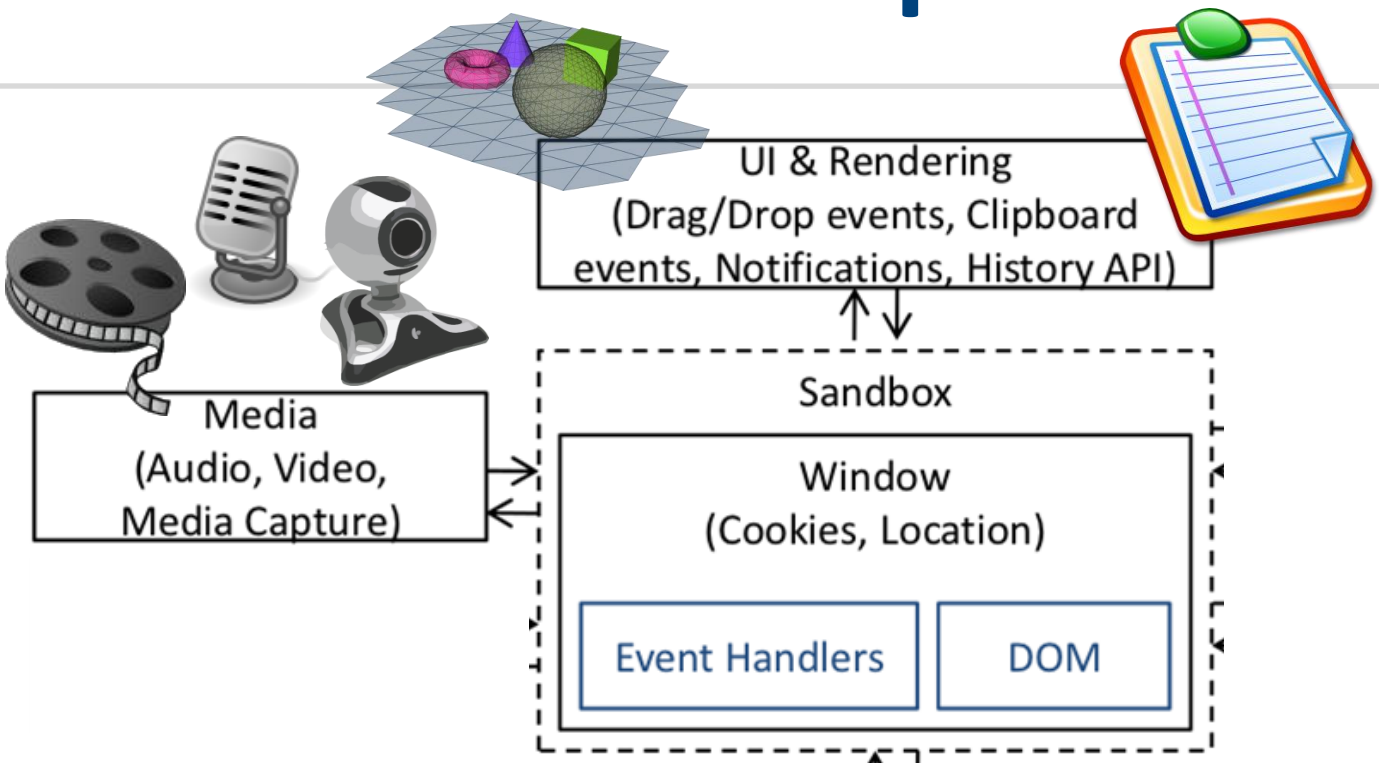
HTML 5 sensitive operations



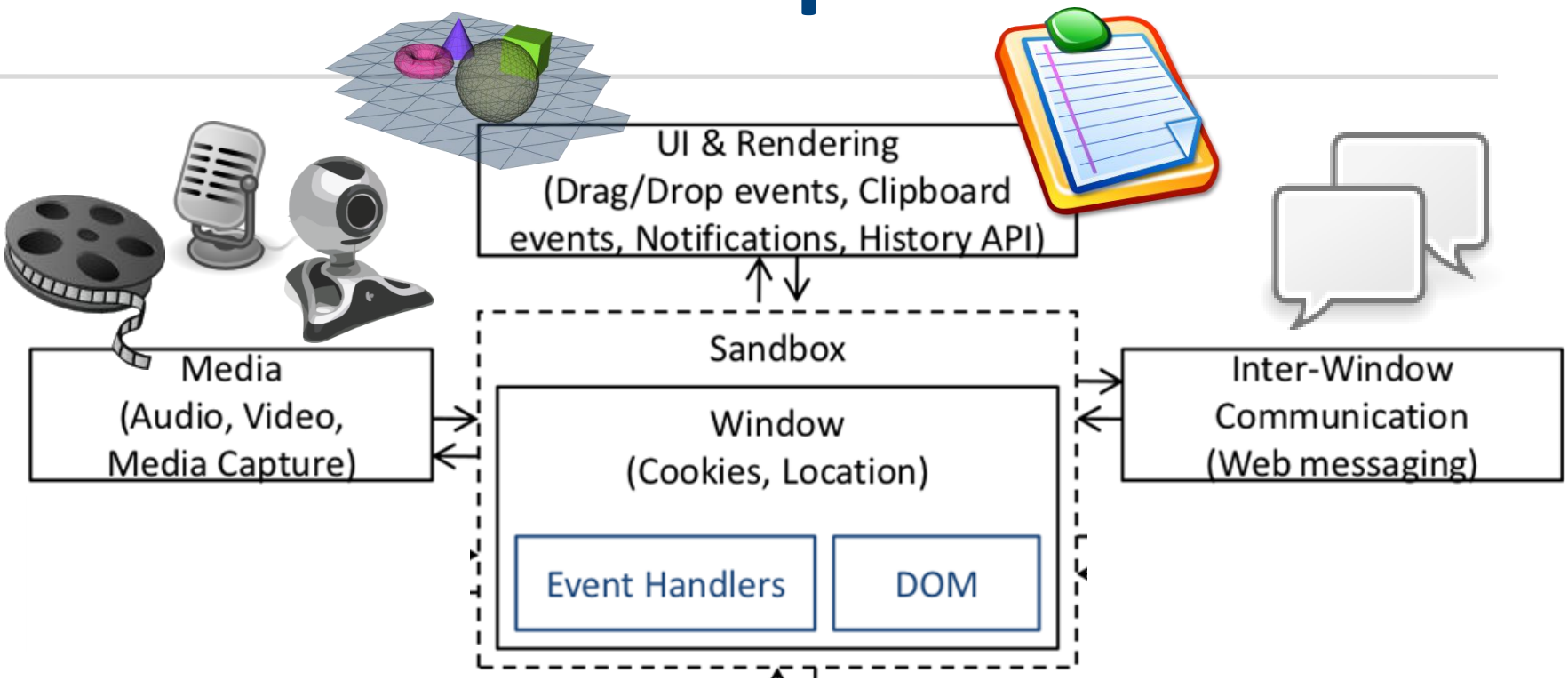
HTML 5 sensitive operations



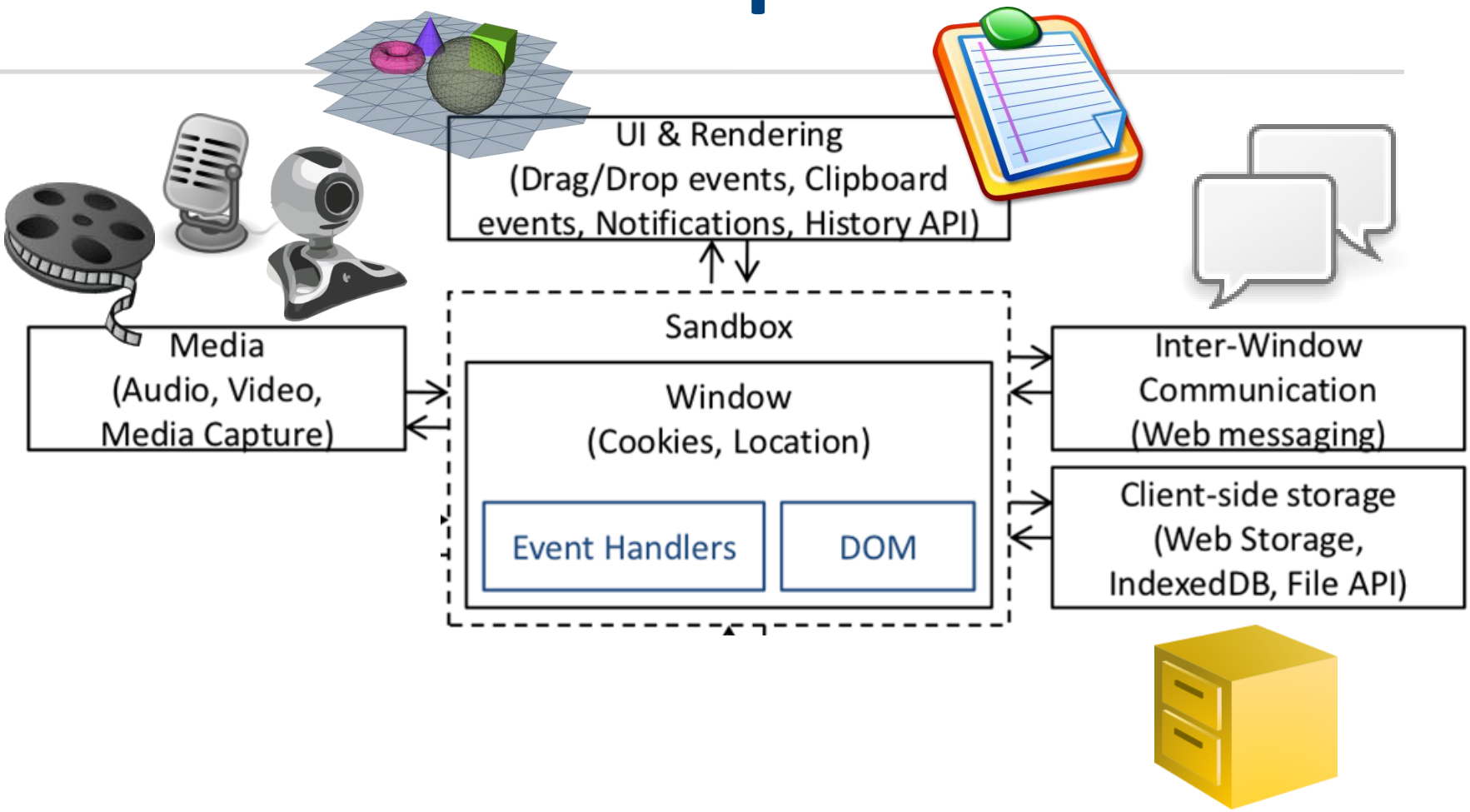
HTML 5 sensitive operations



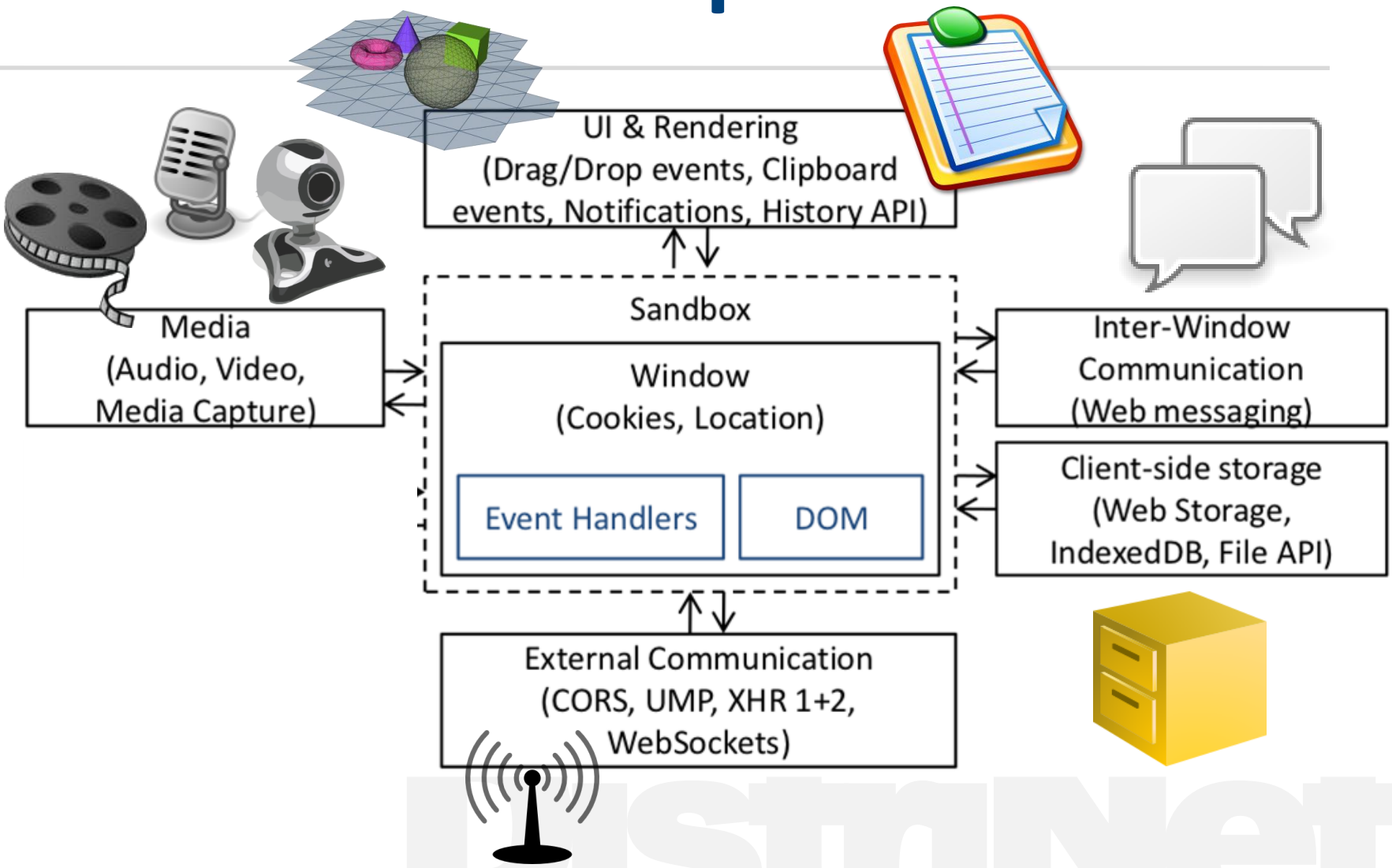
HTML 5 sensitive operations



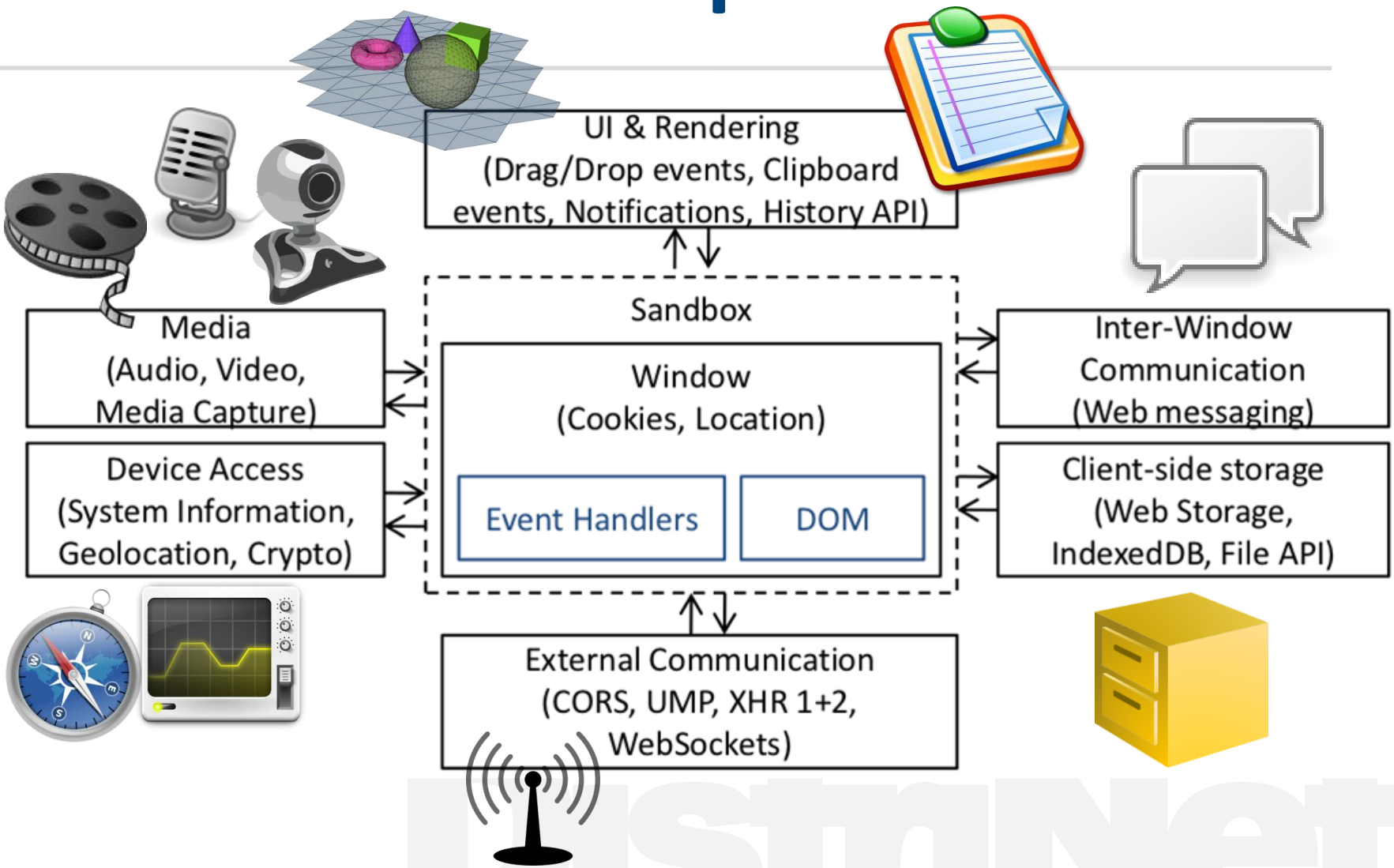
HTML 5 sensitive operations



HTML 5 sensitive operations



HTML 5 sensitive operations



Existing solutions and future trends

Where are we, what's behind us and what's on the horizon?



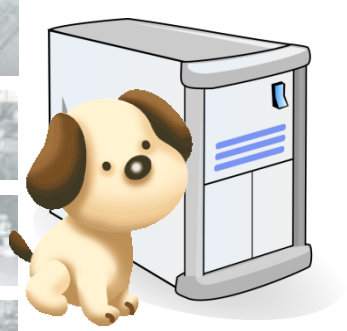



Where to fix the problem?

3. Working with existing tools

2. Modifying the browser

1. Modifying or restricting third party code

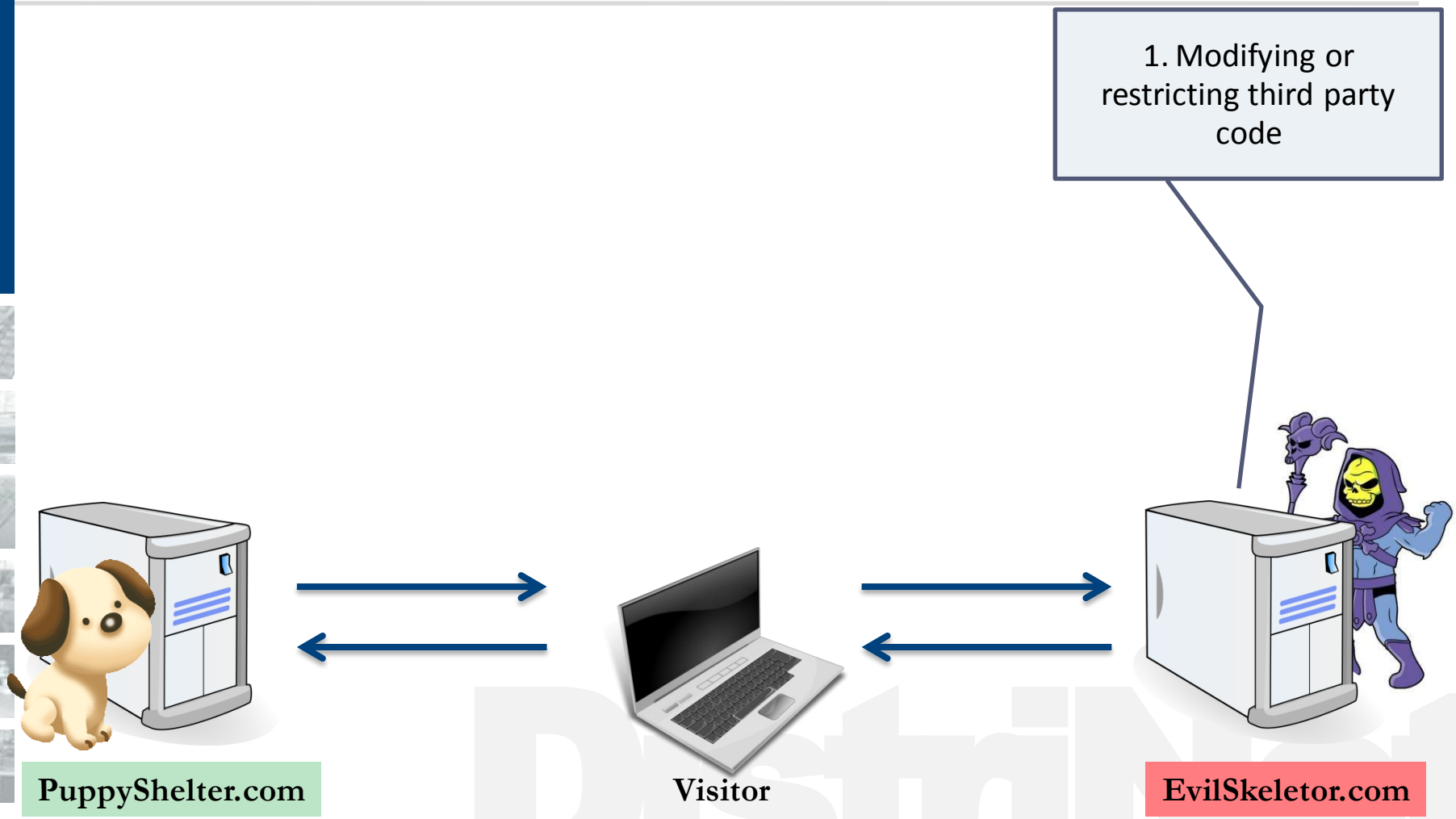


PuppyShelter.com

Visitor

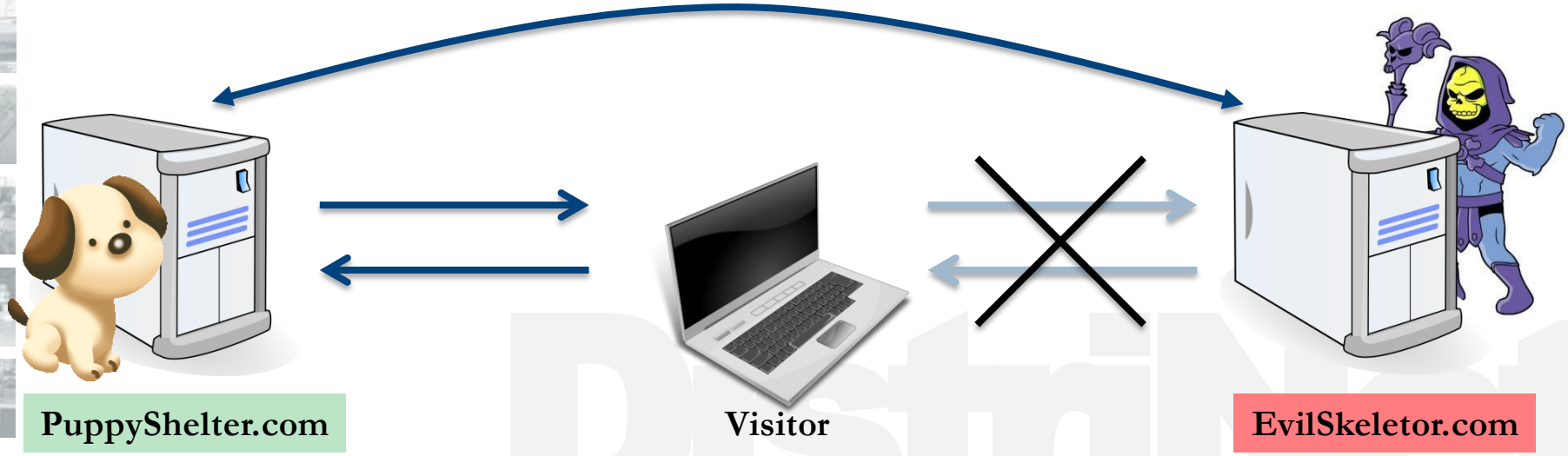
EvilSkeletor.com

Where to fix the problem?



Modifying/restricting 3rd party code

- Tackle the problem at the source
- No direct communication, but through a proxy
- Examples: **Caja**, FBJS, BrowserShield



JavaScript subsets: Caja






Caja: JavaScript subset

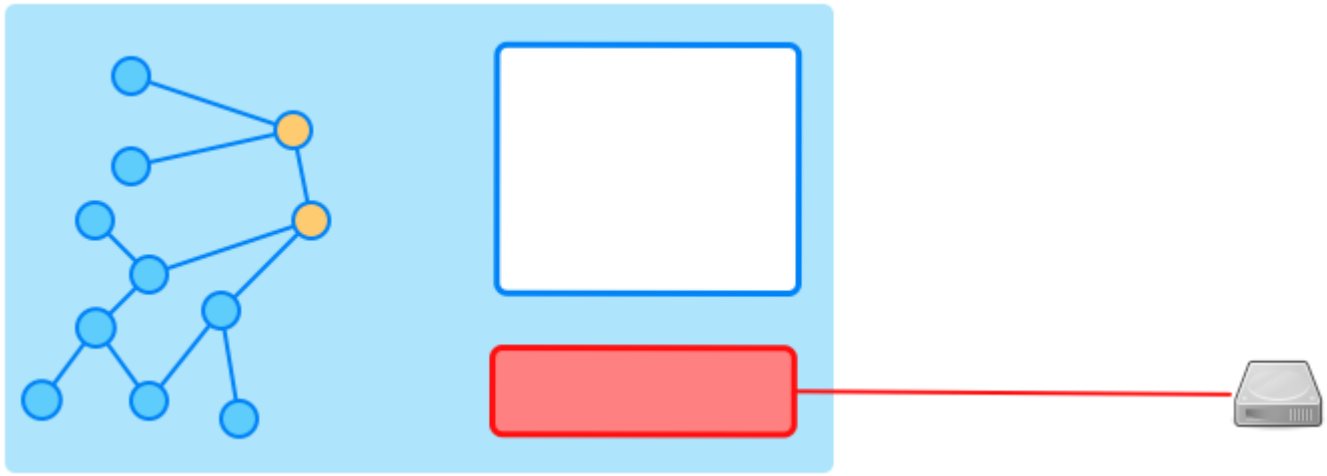
In JavaScript, there is a beautiful, elegant, highly expressive language that is buried under a steaming pile of good intentions and blunders.

-- Douglas Crockford

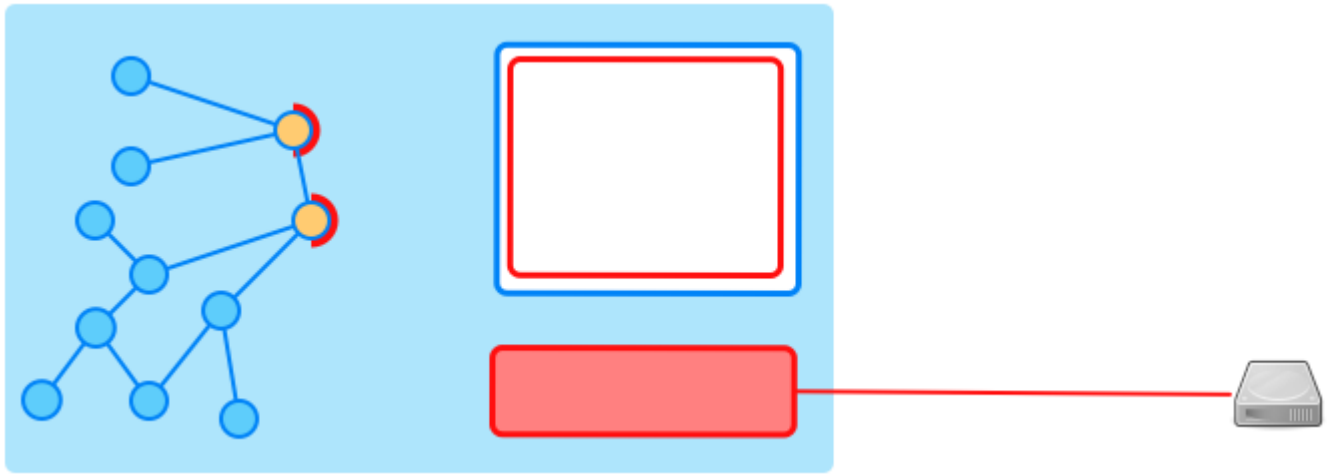
Caja: Capability JavaScript

- Object capability model
  - Functionality is encapsulated in objects
  - Must have a reference to an object to use its functionality
  - No reference means no access
- Caja enforces an object capability model on a safe subset of JavaScript

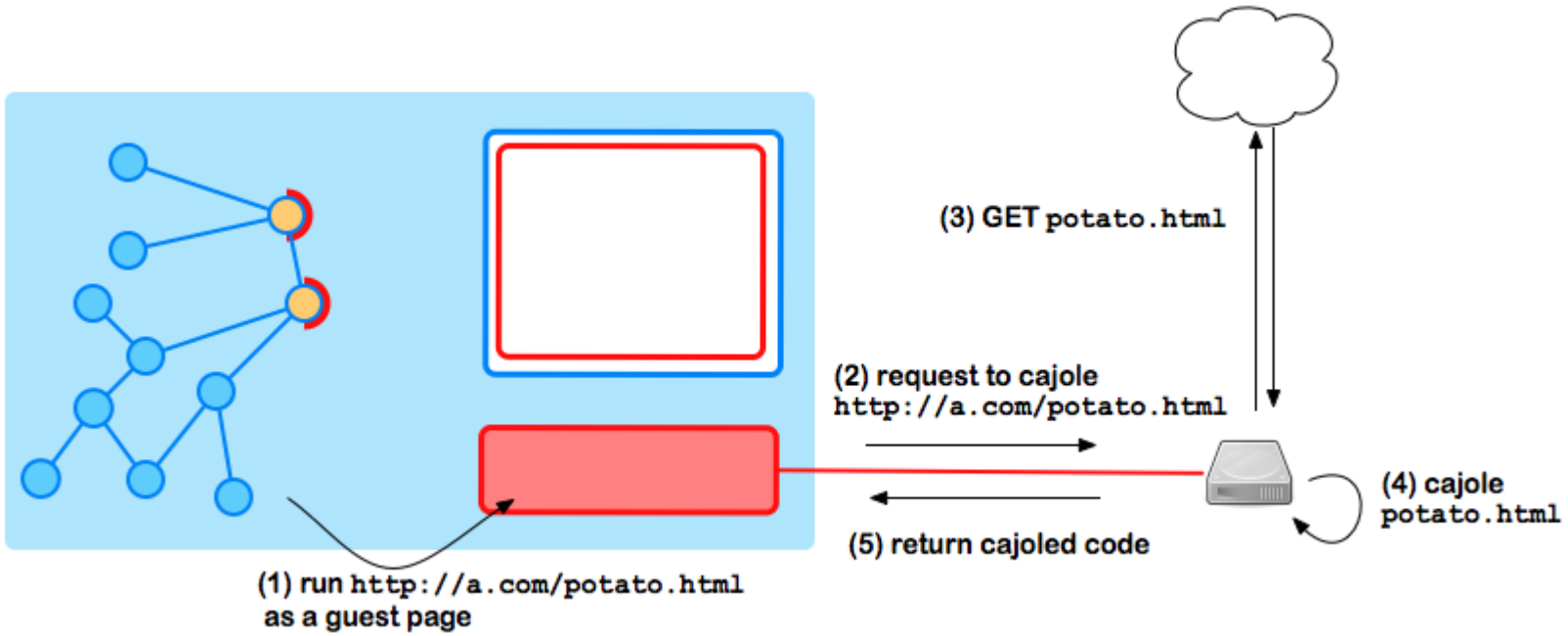
Caja



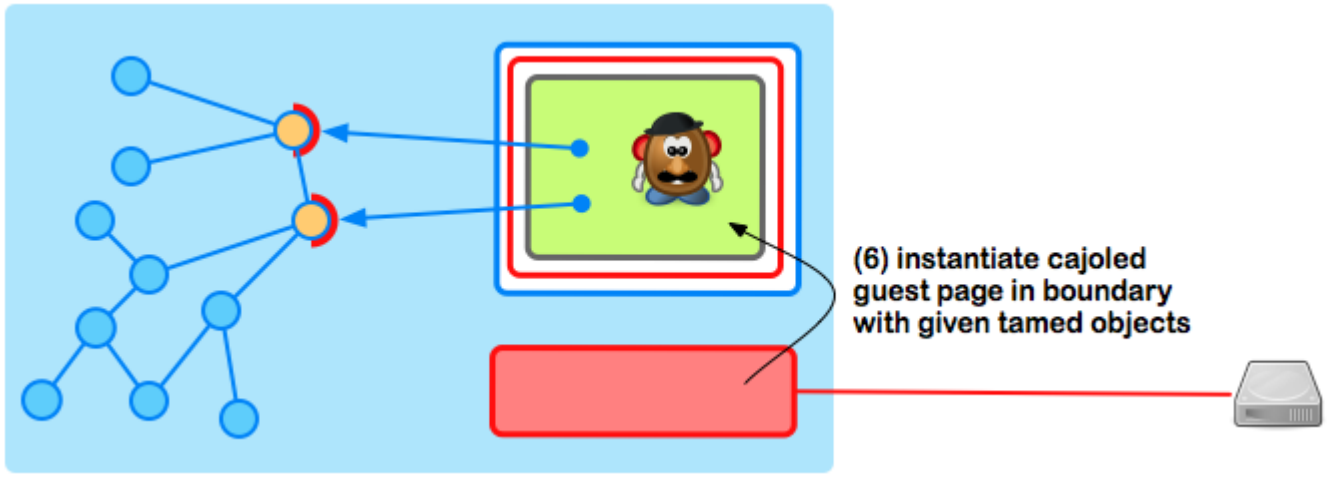
Caja



Caja


(3) GET potato.html
(2) request to cajole
http://a.com/potato.html
(4) cajole
potato.html
(5) return cajoled code
(1) run http://a.com/potato.html
as a guest page

Caja


(6) instantiate cajoled
guest page in boundary
with given tamed objects

Modifying/restricting 3rd party code

- Problems:
  - JavaScript is notoriously difficult to analyze or verify
  - Third party code authors can/will not fit their code into a subset (e.g. eval, with, …)
  - Because of the proxy: SOP issues
    - Existing thirdparty sessions become useless
    - Accessing thirdparty API through e.g. XHR is troublesome

Where to fix the problem?

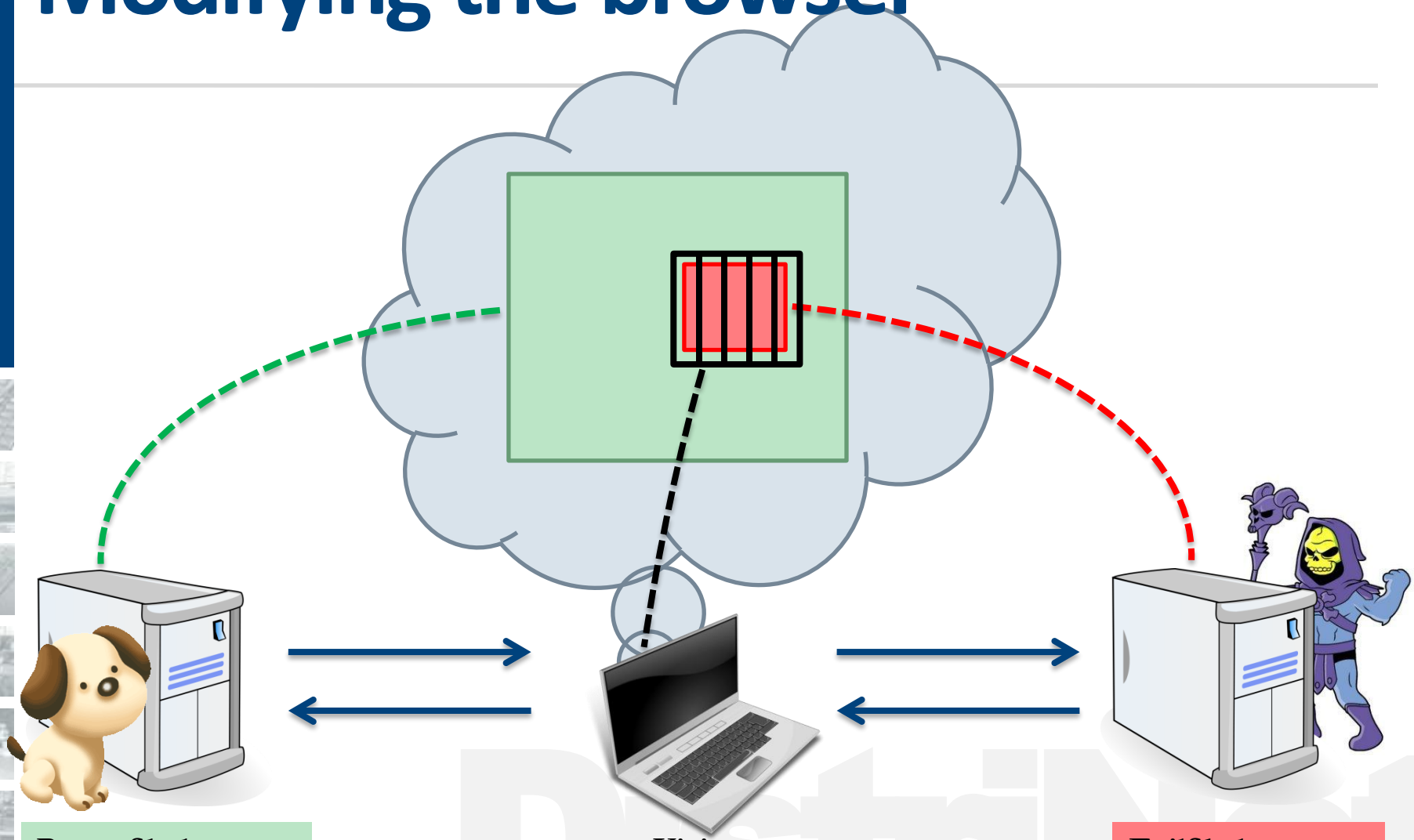
2. Modifying the browser

PuppyShelter.com

Visitor

EvilSkeletor.com

Modifying the browser



Modifying the browser

- Tackle the problem where it manifests itself
  - No need for a JavaScript subset
  - No problems with SOP
- Examples: ConScript, **WebJail**, AdSentry

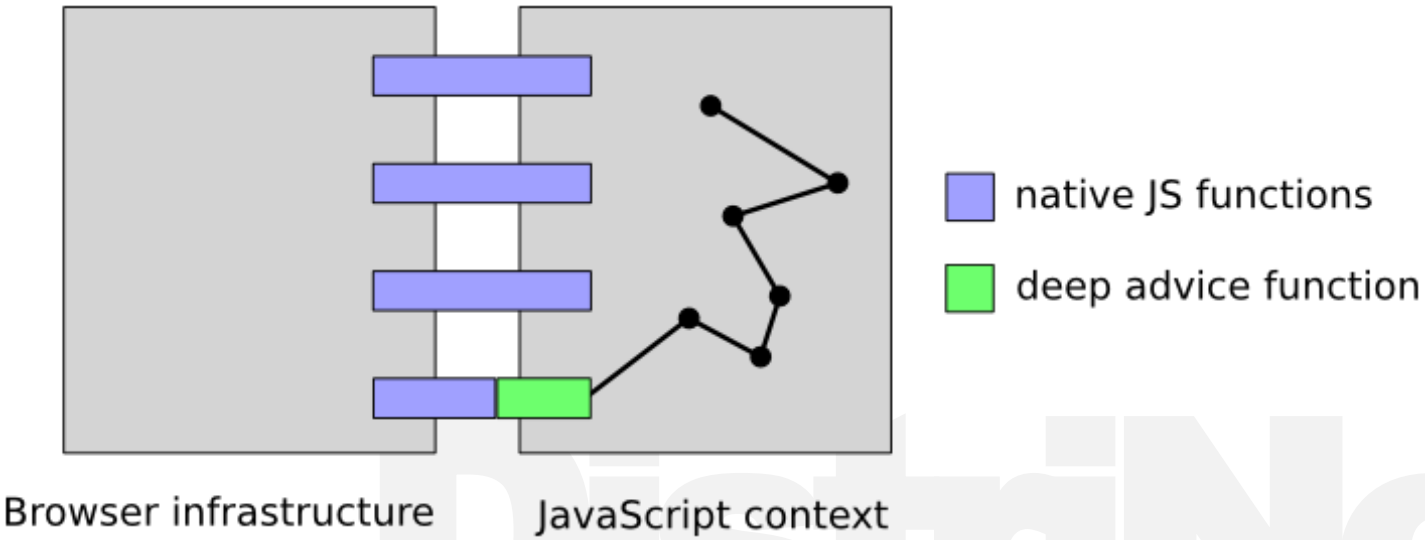
Browser modification: WebJail



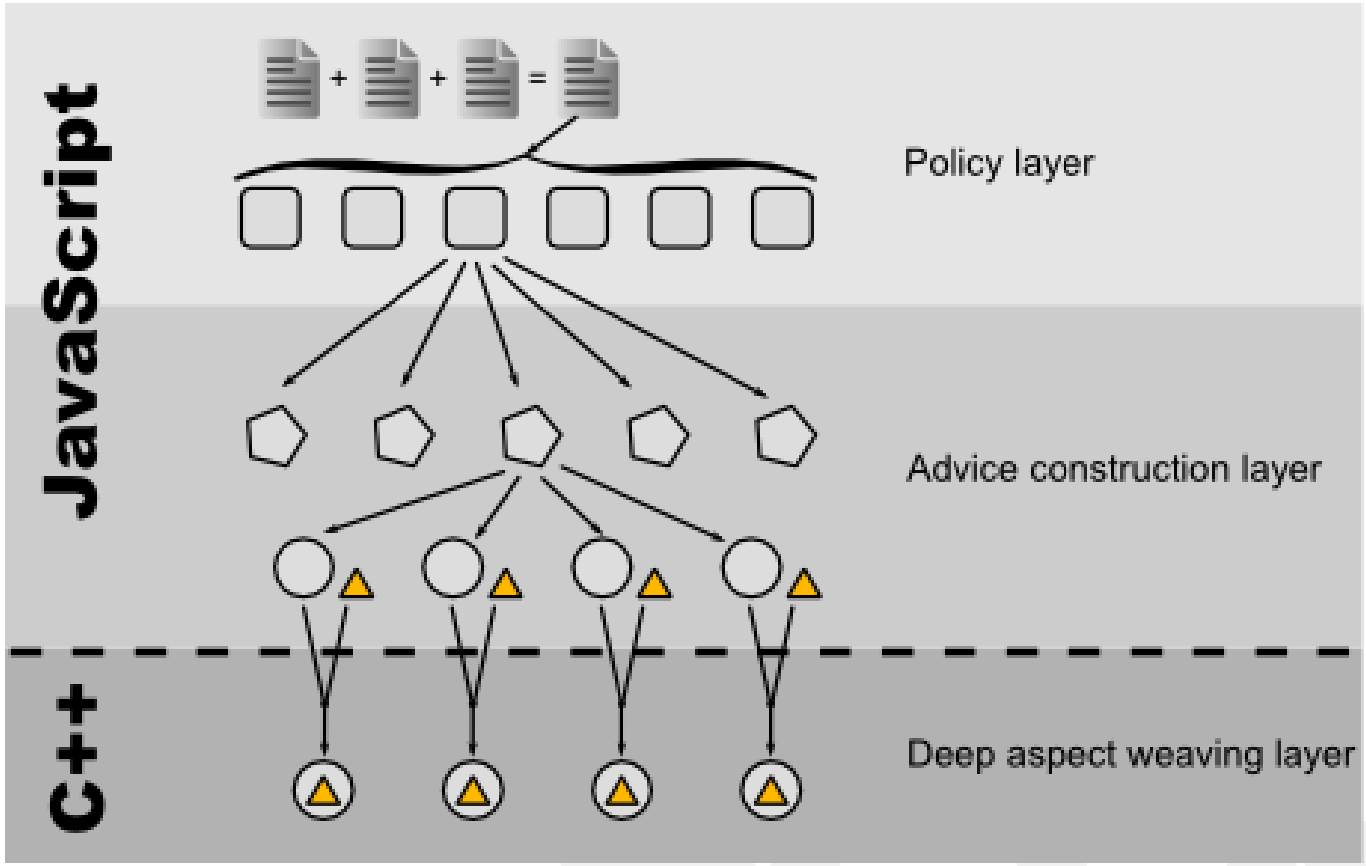


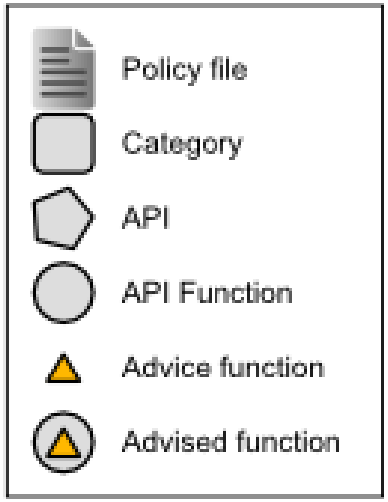

WebJail: Deep aspect weaving layer

- Inspired by ConScript (Meyerovich & Livshits)
- An advice function mediates access for a function
- All access-paths go through the advice function
- Enforced in the browser, advice is locked away safely



WebJail: architecture



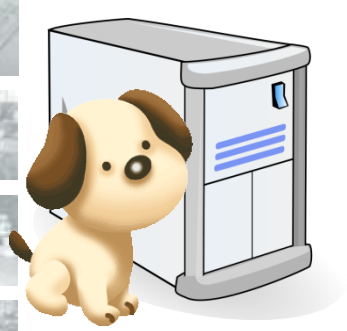


Modifying the browser

- Problem:
  - Deploying a browser modification to all browsers on the internet is hard
- "Just get the modification adopted by W3C so all browsers implement it"

Where to fix the problem?

3. Working with existing tools



PuppyShelter.com

Visitor

EvilSkeletor.com

Use what's available

- HTML5 and ECMAScript5 provide new and powerful functionality
  - → Have been/will be adopted by all browsers
- This is the future
- Examples: **CSP+iframe sandbox, JSand, Treehouse**

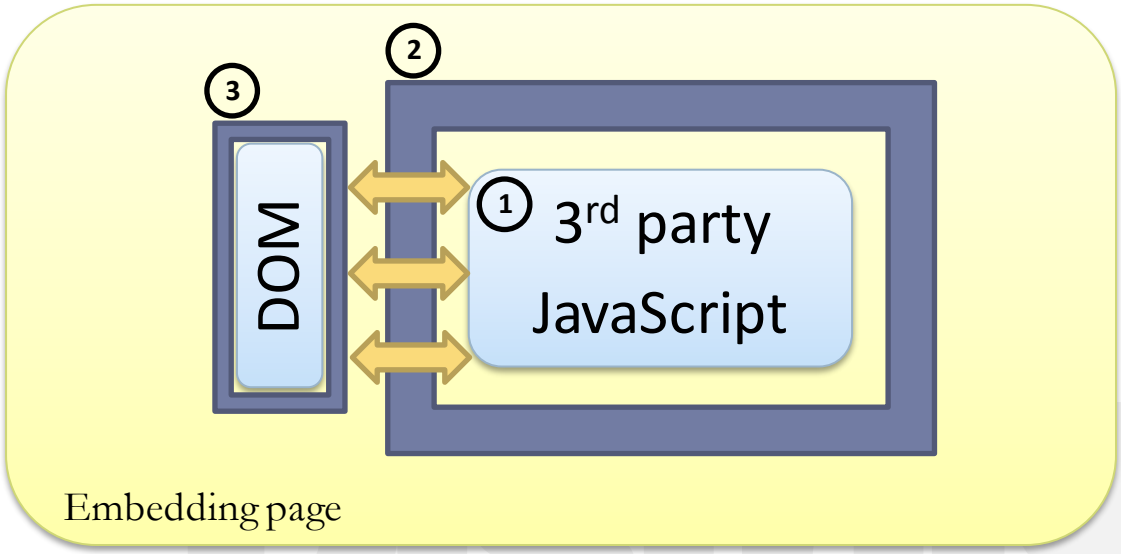
Without modifications: CSP+iframe sandbox, JSand, TreeHouse






Generic modus operandi

1) Download third-party script directly to browser
2) Load script in **isolated** environment
3) Enable controlled access to outside
   Policy determines permitted operations



Content-Security-Policy

- Content-Security-Policy (CSP)
  - Browser-enforced limitation of the resources used by a web application (e.g. use JavaScript only from my own origin)
  - No inline JavaScript allowed
  - Has a report mode
  - Supported by most browsers (in W3C)

Iframe sandbox attribute

- Iframe sandbox attribute
  - Can load and isolate content within a unique origin
  - Can disable all active content like JavaScript
  - Supported by most browsers (in W3C)

Using CSP+iframe sandbox

Host page with CSP policy

Sandboxed iframe with unique origin

- No inline JavaScript
- JavaScript only from same origin
- myDiv.innerHTML = response

- Full JavaScript allowed

→ Used by Google docs/drive for rendering untrusted content e.g. DOC files

JSand

- Wraps DOM in **direct proxy** using membrane pattern
  - Intercept all calls to the DOM
- JavaScript is executed against this wrapped DOM
  - Using **strict mode**
  - Inside a **with block**

TreeHouse

- Uses **webworkers** to isolate JavaScript
  - → No DOM inside webworker
- Calls to the DOM are forwarded using **postMessage**
- Policy determines access to resources

What can you do today?






What to do today?

- Be aware of your own applications
  - Use CSP in report mode to find out what resources are being used
  - Lock your web application down
- Use Iframe sandboxes
  - Restrict active functionality in iframes with third-party content

Need more?

- Some production solutions are available today
- As HTML5/ECMAScript5 are adopted, more solutions will emerge

Thank you!

Questions? Answers!






Acknowledgements

- The work is partially funded by the European FP7 projects WebSand, STREWS and NESSoS.

- With the financial support from the Prevention of and Fight against Crime Programme of the European Union.

b·ccentre
BELGIAN CYBERCRIME CENTRE OF EXCELLENCE FOR TRAINING, RESEARCH & EDUCATION

KU LEUVEN
DISTRINET RESEARCH GROUP

iMinds
CONNECT.INNOVATE.CREATE

References

- Nikiforakis et al. **"You Are What You Include: Large-scale evaluation of remote JavaScript Inclusions"**, CCS 2012
- Van Acker et al. **"WebJail: Least-privilege integration of third-party components in web mashups"**, ACSAC 2011
- Agten et al. **"JSand: Complete client-side sandboxing of third-party JavaScript without browser modifications"**, ACSAC 2012
- Ingram et al. **"TreeHouse: JavaScript sandboxes to help Web developers help themselves"**, ATC 2012
- Reis et al. **"BrowserShield: vulnerability-driven filtering of dynamic HTML"**, OSDI 2006
- Meyerovich et al. **"ConScript: Specifying and Enforcing Fine-Grained Security Policies for JavaScript in the Browser"**, S&P 2010
- Xinshu Dong et al. **"AdSentry: comprehensive and flexible confinement of JavaScript-based advertisements"**, ACSAC 2011

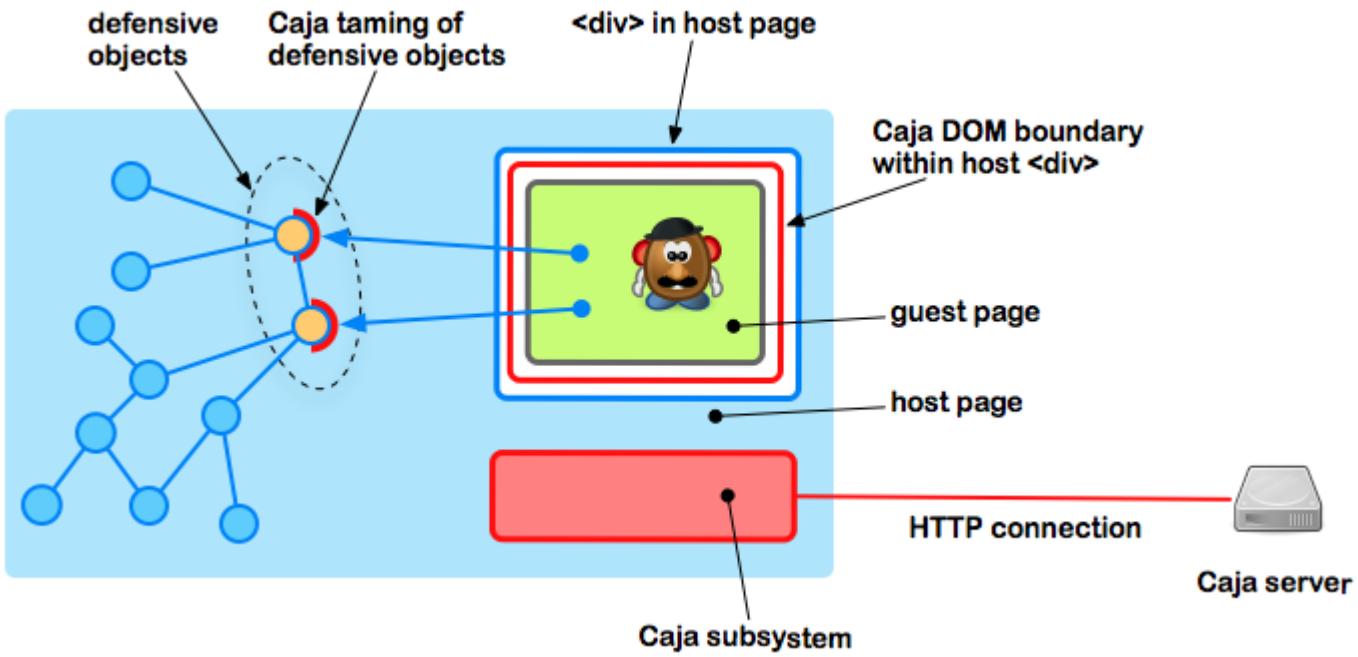
- Caja, https://code.google.com/p/google-caja/
- Mike west, **"Securing the client side: Building safe web applications with HTML5"**, http://parleys.com/#st=5&id=3521
- Content Security Policy, http://www.w3.org/TR/CSP/
- Iframe sandbox attribute, http://www.whatwg.org/specs/web-apps/current-work/multipage/the-iframe-element.html
- Webworkers, http://www.whatwg.org/specs/web-apps/current-work/multipage/workers.html

Appendix: Caja






Caja


defensive objects
Caja taming of defensive objects
 in host page
Caja DOM boundary within host
guest page
host page
HTTP connection
Caja server
Caja subsystem

Appendix: WebJail

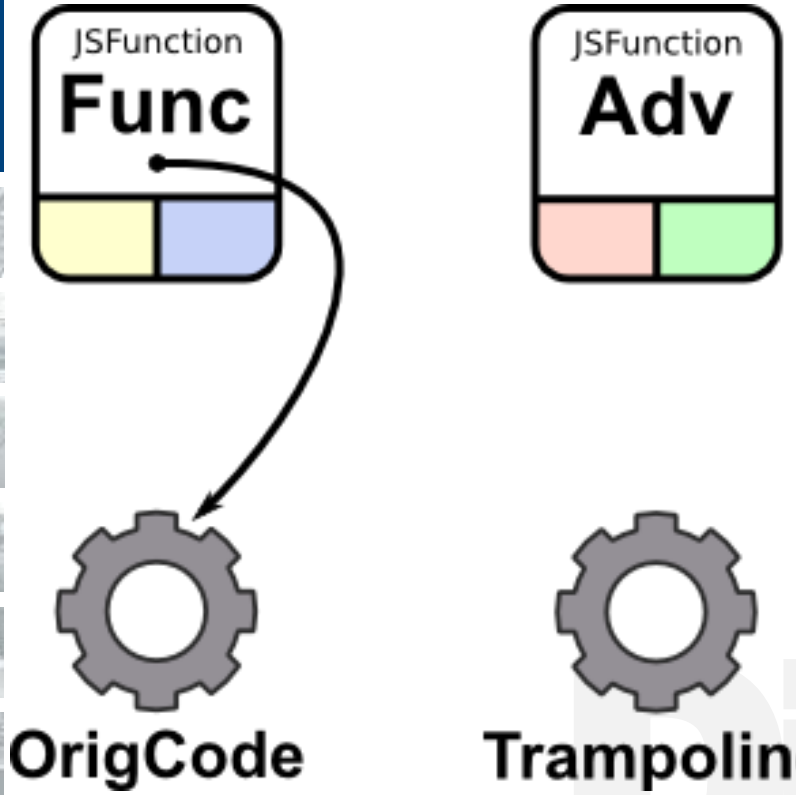




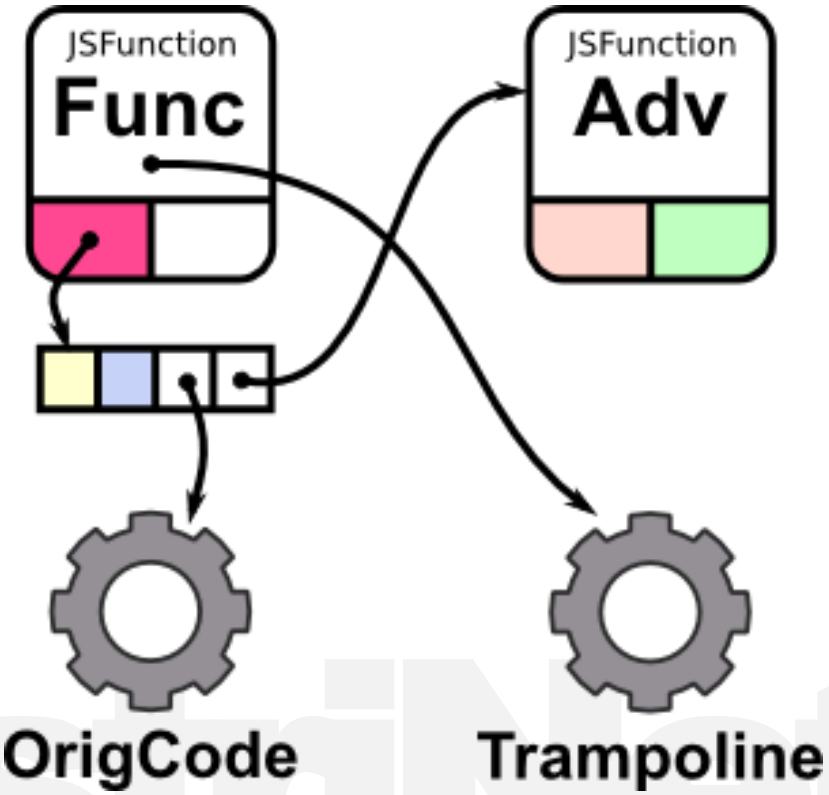

WebJail: Firefox 4.0b7 implementation

(Here be dragons!)

Before advice registration



After advice registration



WebJail: Advice construction: example

```
function makeAdvice(whitelist) {
  var myWhitelist = whitelist;

  return function(origf, obj, vp) {
    if(myWhitelist.indexOf(vp[0])>=0) {
      return origf.apply(obj, vp);
    } else { return false; }
  };
}

adviceFunction = makeAdvice(["hello world", "test"]);
registerAdvice(window.alert, adviceFunction);
```

WebJail: Evaluation

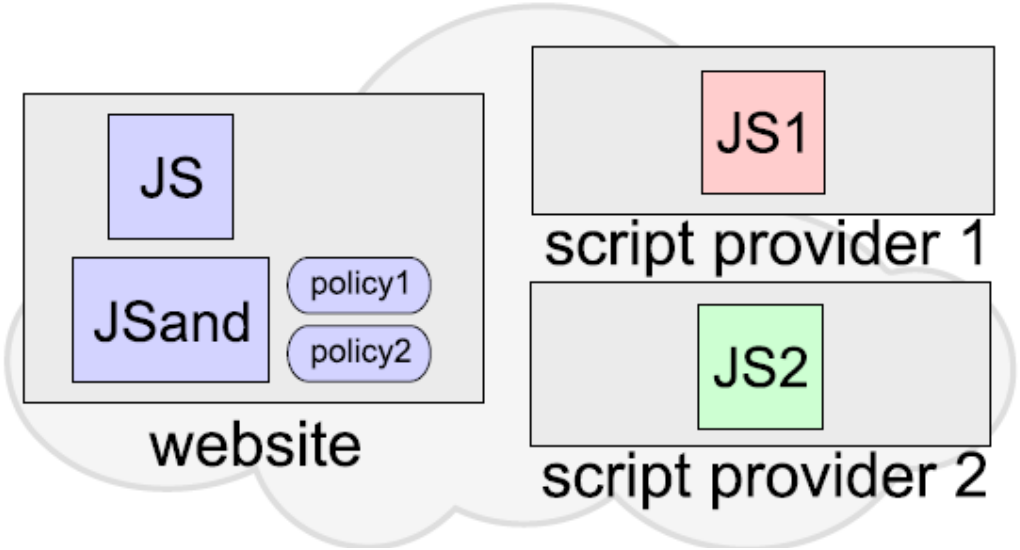
- Performance:
  - Page load-time overhead: 7ms
  - Function execution overhead: 0.1ms
- Security:
  - Manually inspected that all accesspaths are mediated
  - Manually inspected that WebJail infrastructure code is safe
- Applicability
  - Injected policies into real-world iframes using a proxy
  - Facebook application and iGoogle widget: both behave as expected

Appendix: JSand





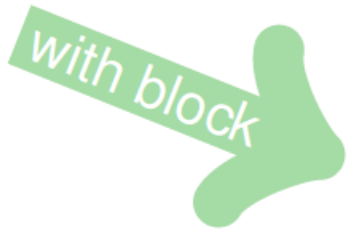

JSand: high-level architecture



JSand: SES example

Secure ECMAScript library usage (simplified):

```
var scriptCode = "window.alert('Boo!');";
ses.execute(scriptCode);
```




```
var catchAll = makeCatchAll();

with (catchAll) {
  (function() {
    "use strict";
    // ** Example malicious code **
    window.alert('Boo!');
    // ****************************
  })();
}
```

JSand: wrapper proxy example

Example (highly simplified):

```
function wrap(target, policy) {
  var handler = {
    get: function(propertyName) {
      if (policy.isPropertyAllowed(target, propertyName)) {
        return wrap(target[name], policy);
      }
      return undefined;
    }
  }
  return Proxy.create(handler);
}

var windowProxy = wrap(window, somePolicy);
windowProxy.alert("Foo");
```

JSand: Performance benchmarks

- Micro benchmarks
  - JSand loadtime: 48.5 ms
  - JQuery loadtime: 1350.6 ms
    - Mainly due to AST script rewriter
    - JQuery loadtime (w/o AST trans): 598.2 ms
  - Membrane transition cost: 7.1 µs
- Macro benchmarks
  - Google Maps loadtime: 1432.8 ms
    - vs 308.0 ms outside JSand
  - Google Maps interaction delay: 420.0 ms
    - vs 320.2 ms outside JSand