Protecting Users by Confining JavaScript with COWL

Deian Stefan, Edward Z. Yang, Petr Marchenko, Alejandro Russo, Dave Herman, Brad Karp, David Mazières
The Web

No longer just a way of publishing static content
The Web

Now app platform; lot of client-side functionality

Core reason: Easy to create complex client-side apps

➤ Combine code and data from different parties!
Many apps handle sensitive data

Political views

Finances

Location info
What do browsers do to ensure that the weather site cannot access my bank statements?
In the beginning: Same-origin Policy

**Idea:** isolate content from different origins

- Compartmentalize code into contexts (tabs, iframes,...)
- Disallow cross-origin reads from contexts & servers
Idea: isolate content from different origins

- Compartmentalize code into contexts (tabs, iframes,...)
- Disallow cross-origin reads from contexts & servers
In the beginning: Same-origin Policy

**Idea:** isolate content from different origins

- Compartmentalize code into contexts (tabs, iframes,...)
- Disallow cross-origin reads from contexts & servers
Problems with SOP

Not strict enough:
can disclose data arbitrarily
  ➤ Third-party code can leak data
  ➤ Code runs with authority of page

Not flexible enough:
can’t read cross-origin data
  ➤ No secure third-party mashups!
Today: SOP + CSP + CORS

Content Security Policy:
➤ Whitelist origins page can communicate with

Cross-origin Resource Sharing:
➤ Server whitelists origins allowed to read the data
Today: SOP + CSP + CORS

Content Security Policy:
- Whitelist origins page can communicate with

Cross-origin Resource Sharing:
- Server whitelists origins allowed to read the data
DAC is not enough!

Forces choice between functionality and privacy

➤ E.g., mint.com-like client-side third-party mashup

➤ **Privacy:** bank doesn’t give mint.cc access to data

➤ **Functionality:** bank cedes user data to mint.cc
   (or worse: user cedes bank credentials)
DAC is not enough!

*Reality:* we give up privacy for functionality!
DAC is not enough!

Third-party mashups
- chase.com
- hsbc.com
- mint.cc

Mutually distrusting services
- docs.google.com
- eff.org

Tightly-coupled libraries
- chase.com

Libraries with narrow APIs
- sketchy.ru
Third-party code + sensitive data

Challenge: allow untrusted code to compute on data

- E.g., chase wants to use password-strength checker library needs to fetch list of common passwords
  - Safe to fetch list before looking at password!

Need: confinement (MAC)

- Impose restrictions on how code uses data
Third-party code + sensitive data

**Challenge**: allow untrusted code to compute on data

- E.g., chase wants to use password-strength checker library needs to fetch list of common passwords
  - Safe to fetch list before looking at password!

**Need**: confinement (MAC)

- Impose restrictions on how code uses data
Third-party code + sensitive data

Challenge: allow untrusted code to compute on data

➤ E.g., chase wants to use password-strength checker library needs to fetch list of common passwords
  - Safe to fetch list before looking at password!

Need: confinement (MAC)

➤ Impose restrictions on how code uses data
Third-party code + sensitive data

Challenge: allow untrusted code to compute on data

➤ E.g., chase wants to use password-strength checker library needs to fetch list of common passwords
  - Safe to fetch list before looking at password!

Need: confinement (MAC)

➤ Impose restrictions on how code uses data
Isn’t confinement a solved problem?

Confinement for Haskell ➞ Hails

Confinement for Java ➞ Jif!

Change JavaScript to enforce IFC with JSFlow
Dev...
Design constraints

• Can’t expect developers to learn new language

• Can’t touch JavaScript runtime
  ➤ Highly optimized JITs
  ➤ Add 1 instruction on hot path ➨ no upstream!

• Can’t radically change the security model
  ➤ Ingrained notion of principals: origins
  ➤ Keep iframes, pages, etc. as security boundaries
The good news

By accident...

Web turns out to be a good fit for confinement

...if you just look at it right
The good news

- Browsers already offer execution contexts
  - Isolation enforced across context boundaries
- Can enforce MAC at context granularity
  - No need to change language runtime! [BFlow]
- Can easily add new DOM-level APIs
  - Attach policies to messages [Hails]
Confinement with Origin Web Labels (COWL)

Key (old) concepts: expressed in practical way?

1. **Labels**: using origins to specify MAC policies
2. **Labeled communication**: security across contexts
   - Avoid changing existing communication APIs
3. **Privileges**: using origins to manage trust
Labels

• Every piece of data is protected by a label

• Label specifies, in terms of origin(s), who cares about the data

➢ E.g., data sensitive to Chase: \texttt{Label(“chase.com”)}

➢ E.g., data sensitive to both Chase and HSBC: \texttt{Label(“chase.com”).and(“hsbc.com”)}
Label tracking

- COWL tracks labels at context/server granularity
  - Pages, iframes, workers, servers

- Messages can be labeled differently from context
  - Both servers & JavaScript can label messages
    - The right way to share sensitive data!
Labeled Communication

• Browser-server communication must respect labels!
Labeled Communication

- Communication across browser contexts must respect label
Labeled Communication

- Communication across browser contexts must respect label
Labeled Communication

• Communication across browser contexts must respect label
Adjusting labels to read data

• Contexts can adopt more restrictive label
  ➤ I.e., add an origin to its label
  ➤ Can then read data from that origin
  ➤ Give up ability to write to contexts without it
Adjusting labels to read data

- Contexts can adopt more restrictive label
  - i.e., add an origin to its label
  - Can then read data from that origin
  - Give up ability to write to contexts without it
Adjusting labels to read data

- Contexts can adopt more restrictive label
  - I.e., add an origin to its label
  - Can then read data from that origin
  - Give up ability to write to contexts without it
Adjusting labels to read data

- Contexts can adopt more restrictive label
  - I.e., add an origin to its label
  - Can then read data from that origin
  - Give up ability to write to contexts without it
Adjusting labels to read data

- Contexts can adopt more restrictive label
  - I.e., add an origin to its label
  - Can then read data from that origin
  - Give up ability to write to contexts without it
Summary: COWL design

Web was made for confinement

1. Origins are a natural way to specify labels

2. Leverage contexts as security boundaries
   ➤ Mixed-granularity: label messages

3. Use origins to express privileges (see paper)
What can we do with this?
Example: client-side Mint

• Read-only client-side personal finance service

• Banks can make labeled statements available to Mint ➔ Flexibility+Privacy!
Example: client-side Mint

• Read-only client-side personal finance service

• Banks can make labeled statements available to Mint ➔ Flexibility+Privacy!
Example: client-side Mint

- Read-only client-side personal finance service

- Banks can make labeled statements available to Mint ➠ **Flexibility + Privacy!**
Example: client-side Mint

- Read-only client-side personal finance service

- Banks can make labeled statements available to Mint ➠ Flexibility + Privacy!
Example: client-side Mint

- Read-only client-side personal finance service
- Banks can make labeled statements available to Mint ➤ **Flexibility + Privacy!**
Example: client-side Mint

- Read-only client-side personal finance service

- Banks can make labeled statements available to Mint ➔ Flexibility + Privacy!
Example: client-side Mint

- Read-only client-side personal finance service

- Banks can make labeled statements available to Mint ➞ Flexibility+Privacy!
Example: client-side Mint

- Read-only client-side personal finance service

- Banks can make labeled statements available to Mint ➔ **Flexibility + Privacy!**
We built it...
Implementations

• DOM-level API for both Firefox and Chromium
  ➤ No changes to JavaScript engines
  ➤ Maintain existing communication APIs
  ➤ For each page COWL only enabled on first use of API

• Gecko and Blink: roughly 4K lines of C++ each
Evaluation: Performance

- Overhead of securing a mashup service?
- Overhead of compartmentalization?
- Will adding COWL slow the existing Web?
Evaluation: Performance

Worst-case (loopback, trivial app code)
end-to-end page load: roughly 16% [16ms]

For real apps: relative overhead is small!
Evaluation: Applicability

Third-party mashups
- chase.com
- hsbc.com
- mint.cc

Mutually distrusting services
- docs.google.com
- eff.org

Tightly-coupled libraries
- jQuery

Libraries with narrow APIs
- chase.com
- sketchy.ru
Deployability

• High degree of backward compatibility
  ➤ Does not affect pages that do not use COWL API

• Reuse existing concepts (origins, contexts)
  ➤ Expect it to be friendly to developers

• Implementations possible for major browsers
  ➤ Changes don’t touch JavaScript engine
Limitations & future work

• Covert channels
  ➢ Malicious code may still covertly leak data
  ➢ COWL enforces MAC in addition to existing DAC

• Compartmentalization
  ➢ Cannot just label and run existing apps
  ➢ Compartmentalizing applications requires thought
Related work

• Coarse-grained confinement: BFlow
  ➢ Mainly concerned with untrusted code
  ➢ COWL also handles the mutually distrusting case

• Fine-grained confinement: JSFlow
  ➢ Better fit for tightly-coupled libraries
  ➢ New semantics, 100x slowdown
Conclusion

Today: give up privacy for flexibility

➤ Modern web apps need to compute on sensitive data
➤ DAC is crucial, but insufficient!

COWL: confinement for client-side code

➤ Naturally extends the existing web model
➤ Achieves both flexibility and privacy without slowdown
Thanks!

http://cowl.ws