# COWL
## Confinement with Origin Web Labels

Deian Stefan

http://cowl.ws

STANFORD
UNIVERSITY

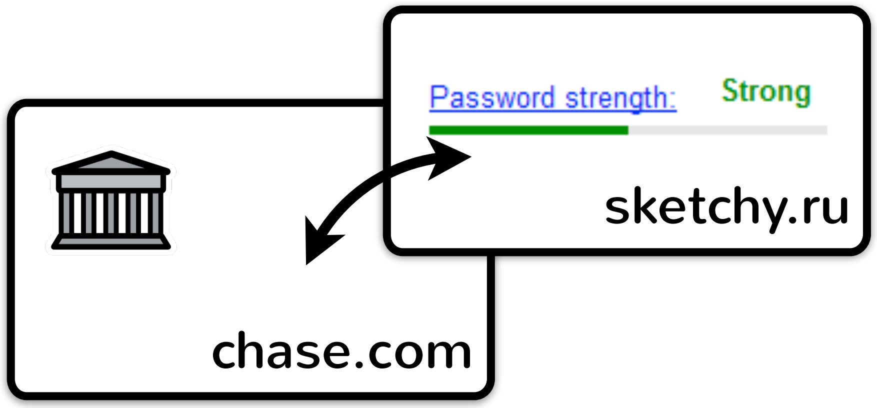
## Where COWL comes from…

$$\left.\begin{array}{r}\textbf{SOP}\\ \textbf{CSP}\\ \textbf{CORS}\end{array}\right\} \begin{array}{l}\text{Discretionary}\\ \text{Access}\\ \text{Control}\end{array}$$

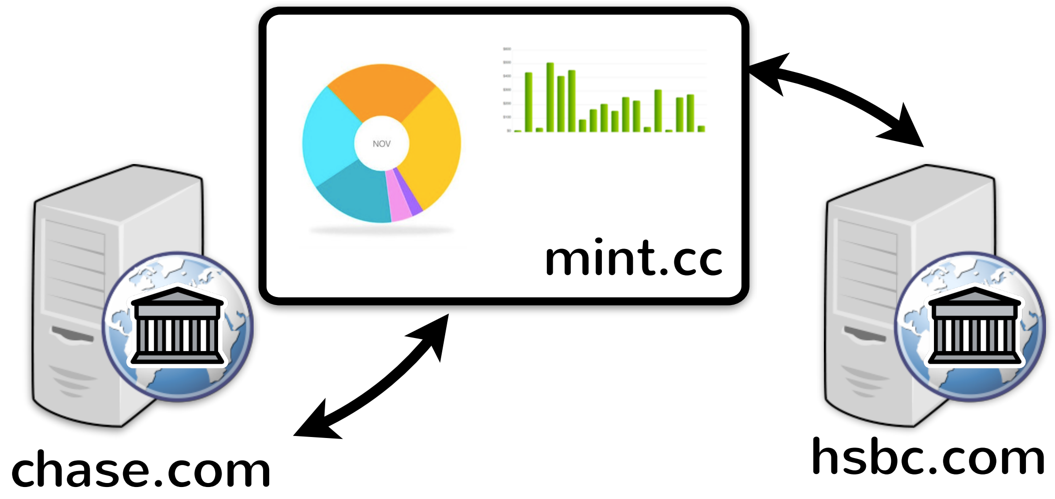
Crucial for the Web, but fall short in some cases…

# Where does DAC fall short?

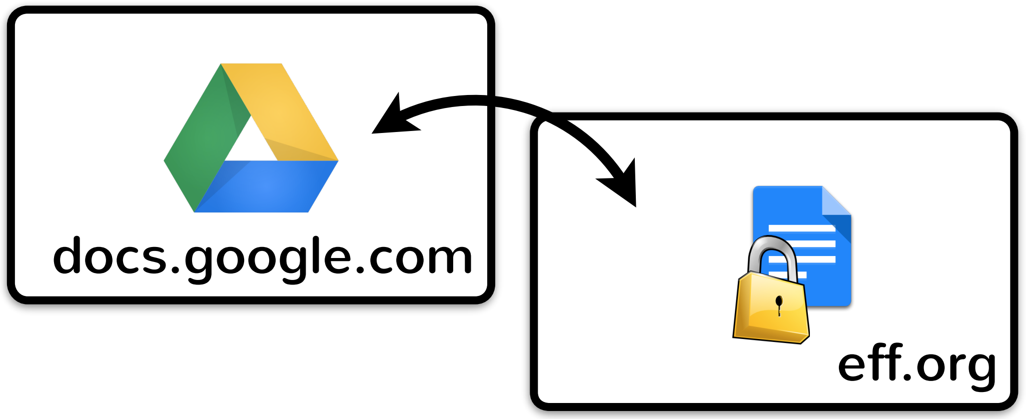
## Untrusted libraries



## Third-party mashups



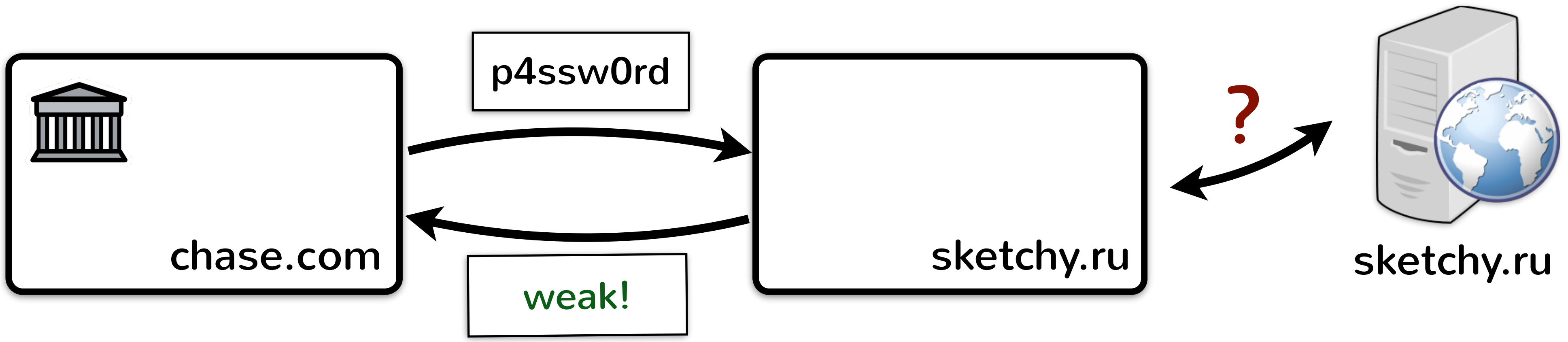
## Mutually distrusting services



# How does DAC fall short?

Forces choice between functionality and privacy

- E.g., password strength checker library

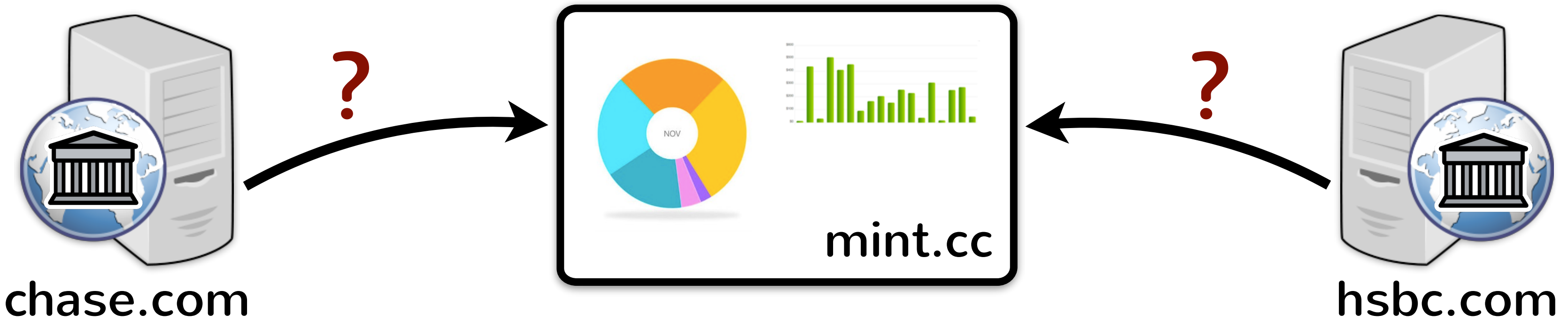


- **Privacy:** use CSP+sandbox to disallow communication
- **Functionality:** allow checker to fetch common pass.

# How does DAC fall short?

**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)

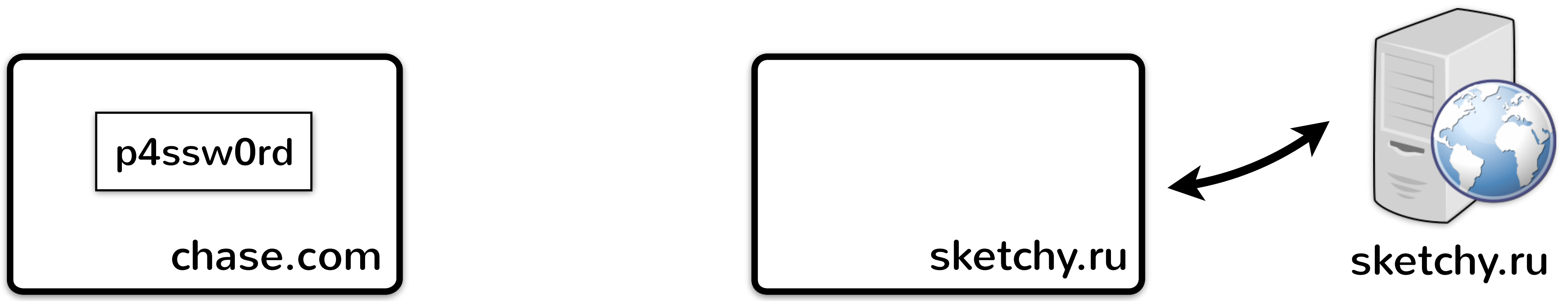
## Why does DAC fall short?

- **Fundamentally**
  - Apps rely on and use third-party code
  - This code computes on sensitive data
- **DAC restricts who can access data**
  - Not what it can do with the data once granted access!

# Confinement (at a glance)

**Idea: impose restrictions on how code uses data**

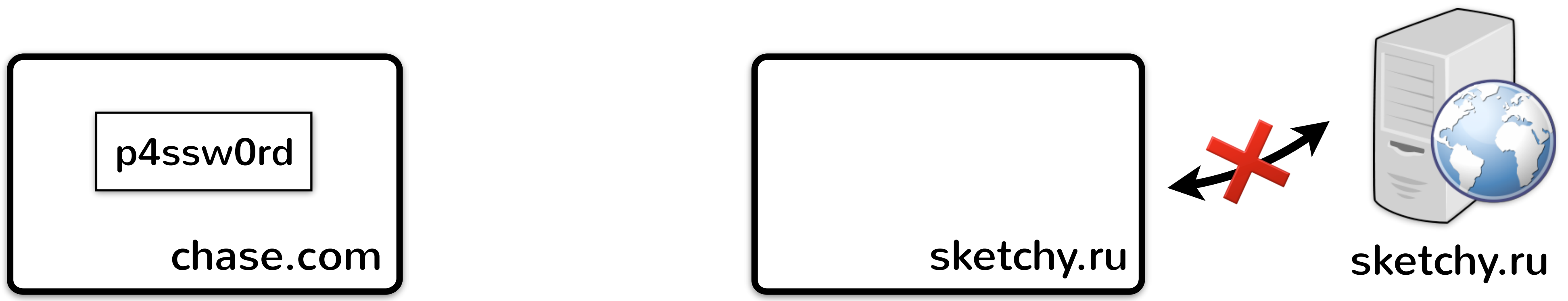
- E.g., it is safe to fetch list of common password before looking at password, but once password is inspected
  - ➠ restrict communication!



# Confinement (at a glance)

**Idea: impose restrictions on how code uses data**

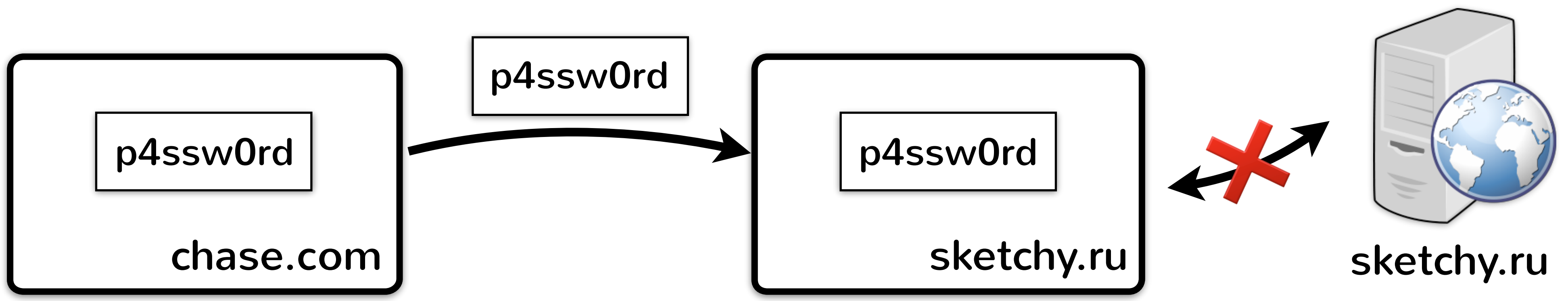
- E.g., it is safe to fetch list of common password before looking at password, but once password is inspected
  - ➠ restrict communication!



# Confinement (at a glance)

**Idea: impose restrictions on how code uses data**

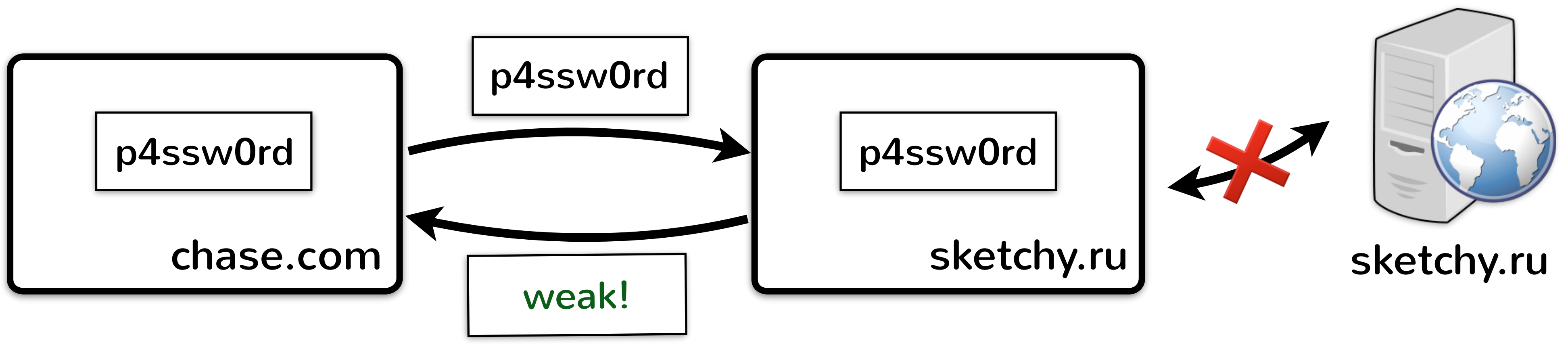
- E.g., it is safe to fetch list of common password before looking at password, but once password is inspected
  - ➠ restrict communication!



# Confinement (at a glance)

**Idea: impose restrictions on how code uses data**

- E.g., it is safe to fetch list of common password before looking at password, but once password is inspected
  - ➠ restrict communication!



## COWL goals

- **Allow authors to declare secrecy/integrity policy on data**
  - ➤ Ensure data is protected when shared with untrusted code (third-party or own)
- **Allow authors to consider own code untrusted**
  - ➤ I.e., allow authors to privilege separate their apps and run code with least privileges

## COWL use cases

- Confining untrusted third-party services
- Sharing data with third-party mashups
- Content isolation (via privilege separation)
  - Similar to suborigin use case
- Running content with least privileges

# COWL framework

- **Policy specification**
  - Labels: generalization of origin to conjunctions and disjunctions of origins
  - Privileges: makes authority of pages explicit with labels
- **Policy enforcement**
  - Enforcement of policy end-to-end via confinement

## Labels

- **Specifying policy in the browser**
  - Labels are associated with contexts
  - Labels can be associated with clonable objects (use with postMessage and XHR)
- **Specifying policy server-side**
  - Labels can be associated with XHR responses

## Privileges

- **Explicit control over page privilege with JS**
  - ➤ Page has default privilege = its origin
- **Controlling defaul page privilege**
  - ➤ E.g., to ensure page doesn’t have authority of origin

## Confinement enforcement

- **Each context has a COWL state**
  - Labels and privilege of context used to dictate who context can communicate with
- **In browser: changes to WebMessaging & HTML5**
- **Browser-server: changes to Fetch**
  - Restrict requests to prevent leaking browser data
  - Restrict responses if label of response is more sensitive than context label (non labeled-json request)

## Current status

- **FPWD**
  - Need to address some issues, but most features in place
- **Implementation**
  - Firefox patch in the works, should be done this year
  - Working on polyfill (using CSP & 3rd trusted server)
  - Would love to talk to Chromium/IE teams

## Issues

- What to do about same-origin, but differently labeled contexts
- Overtainting in top-level context
  - Allow leak? Leave it open-ended?
- Warnings and security reporting: what to do about bad headers

# Future directions

- **Light weight, in-context workers**
  - Make it easier to compartmentalize apps
- **Labeled-credentials?**
- **Clearance**
  - Useful way to ensure context can never read certain kinds of data
- **Covert channels**
  - Enumerate leakage points (maybe as separate doc?)

## Thanks!

`http://cowl.ws`