

Modern client-side defenses

Deian Stefan

Modern web "site"

The screenshot shows a browser window displaying the New York Times website. The address bar shows the URL: www.nytimes.com/pages/nyregion/index.html?module=HPMiniNav&contentCollection=New York&pgtype=Homepage®ion=TopBar&action=click&t. The page features a navigation bar with the New York Times logo, a search icon, and the text "N.Y. / Region". There are buttons for "SUBSCRIBE NOW", "SIGN IN", and "Register".

A large advertisement for Volkswagen is displayed, featuring a red car and the text: "Lease a 2014 Golf TDI for \$289/mo for 36 months. \$0 Down". The Volkswagen logo and "Das Auto." are also visible.

The main content area includes an article titled "In New York, Hard Choices on Health Exchange Spell Success" by ANEMONA HARTOCOLLIS, published 49 minutes ago. The article is accompanied by a photo of a woman at a doctor's appointment. Below the photo is a caption: "Chang W. Lee/The New York Times".

The article text reads: "Malka Percal, 63, had her blood pressure checked and got an EKG test at a doctor's appointment. Her co-pay was \$75. More than 900,000 residents signed up for health plans, and premiums have dropped, though the state limited consumers' choices."

Other content on the page includes a "SIDE STREET" section titled "An Artist Takes His Pay in Coffee and Community" by DAVID GONZALEZ, a "BIG CITY BOOK CLUB" section, and a "FASHION" section titled "Intersection: Elmhurst Style Ease". There is also a "VIDEO" section and a "TWITTER" section for @NYTMETRO.

Modern web "site"

The screenshot displays the New York Times website's 'N.Y. / Region' section. At the top, the navigation bar includes the New York Times logo, the page title 'N.Y. / Region', and buttons for 'SUBSCRIBE NOW', 'SIGN IN', and 'Register'. Below the navigation bar is a large advertisement for Volkswagen's 'TDI CleanDiesel Event', featuring a red car and the text 'Lease a 2014 Golf TDI for \$289/mo for 36 months. \$0 Down'. The main content area is divided into several sections:

- Health Article:** 'In New York, Hard Choices on Health Exchange Spell Success' by ANEMONA HARTOCOLLIS, 49 Minutes Ago. The article includes a photo of a woman at a doctor's appointment and a caption: 'Malka Percal, 63, had her blood pressure checked and got an EKG test at a doctor's appointment. Her co-pay was \$75. More than 900,000 residents signed up for health plans, and premiums have dropped, though the state limited consumers' choices.'
- Side Street Column:** 'An Artist Takes His Pay in Coffee and Community' by DAVID GONZALEZ. The article features a photo of an artist painting a mural and a caption: 'David Ellis, whose other work often fetches tens of thousands of dollars, is giving back to his neighborhood by painting a large mural in a bodega in Fort Greene, Brooklyn. • More Side Street Columns »'
- Big City Book Club:** 'BIG CITY BOOK CLUB' with a photo of a book cover titled 'Brooklyn' and a caption: 'Brooklyn today is a destination for celebrities and wealthy creative types,'
- Twitter Widget:** '@NYTMETRO ON TWITTER' with a 'FOLLOW' button and the text: 'For the latest news, analysis and journalists' perspectives. Twitter list of staff journalists »'
- Video Section:** 'VIDEO »' with a 'More New York Videos »' link and a video player showing a man in a white cap.
- Fashion Article:** 'FASHION Intersection: Elmhurst Style Ease' with a caption: 'In the Queens neighborhood of Elmhurst, locals like to keep their style casual.'

Modern web "site"

Page code



The screenshot displays a modern web page layout for The New York Times, specifically the 'N.Y. / Region' section. The page features a clean, organized design with a prominent navigation bar at the top. The main content area is divided into several columns, each containing different types of content: a large advertisement for Volkswagen, a featured news article with a photo, a social media widget for Twitter, a video player, and smaller text-based articles. The layout is responsive and visually appealing, with clear typography and a consistent color scheme.

Navigation Bar: The New York Times logo, search icon, and 'N.Y. / Region' header. Buttons for 'SUBSCRIBE NOW', 'SIGN IN', and 'Register' are visible.

Advertisement: Volkswagen TDI CleanDiesel Event. Lease a 2014 Golf TDI for \$289/mo for 36 months. \$0 Down. Includes a red car image and the VW logo.

Featured Article:
In New York, Hard Choices on Health Exchange Spell Success
By ANEMONA HARTOCOLLIS 49 Minutes Ago
Chang W. Lee/The New York Times
Malka Percal, 63, had her blood pressure checked and got an EKG test at a doctor's appointment. Her co-pay was \$75. More than 900,000 residents signed up for health plans, and premiums have dropped, though the state limited consumers' choices.

SIDE STREET:
An Artist Takes His Pay in Coffee and Community
By DAVID GONZALEZ
David Ellis, whose other work often fetches tens of thousands of dollars, is giving back to his neighborhood by painting a large mural in a bodega in Fort Greene, Brooklyn.
• More Side Street Columns »

BIG CITY BOOK CLUB:
Brooklyn today is a destination for celebrities and wealthy creative types,
Brooklyn

SOCIAL MEDIA: @NYTMETRO ON TWITTER. For the latest news, analysis and journalists' perspectives. Twitter list of staff journalists » FOLLOW

VIDEO: More New York Videos »

FASHION:
Intersection: Elmhurst Style Ease
In the Queens neighborhood of Elmhurst, locals like to keep their style casual.

Modern web "site"

Page code

Ad code

The screenshot shows a browser window displaying the New York Times website. The address bar shows the URL: `www.nytimes.com/pages/nyregion/...ex.html?module=HPMiniNav&contentCollection=New York&pgtype=Homepage®ion=TopBar&action=click&t`. The page header includes the New York Times logo, the text "N.Y. / Region", and buttons for "SUBSCRIBE NOW", "SIGN IN", and "Register".

A large advertisement for Volkswagen is featured, titled "Volkswagen TDI CleanDiesel Event". It offers a lease for a 2014 Golf TDI for \$289/month for 36 months with \$0 down. The ad includes the Volkswagen logo and the slogan "Das Auto.".

Below the ad, there are several news articles:

- In New York, Hard Choices on Health Exchange Spell Success**
By ANEMONA HARTOCOLLIS 49 Minutes Ago
Chang W. Lee/The New York Times
Malka Percal, 63, had her blood pressure checked and got an EKG test at a doctor's appointment. Her co-pay was \$75. More than 900,000 residents signed up for health plans, and premiums have dropped, though the state limited consumers' choices.
- SIDE STREET: An Artist Takes His Pay in Coffee and Community**
By DAVID GONZALEZ
David Ellis, whose other work often fetches tens of thousands of dollars, is giving back to his neighborhood by painting a large mural in a bodega in Fort Greene, Brooklyn.
• More Side Street Columns »
- BIG CITY BOOK CLUB**
Brooklyn today is a destination for celebrities and wealthy creative types,
Brooklyn

On the right side of the page, there is a Twitter widget for @NYTMETRO ON TWITTER and a video player for "More New York Videos" featuring a man in a white cap.

Modern web "site"

Page code

Ad code

Third-party APIs

The screenshot shows a browser window displaying the New York Times website. The page layout includes a header with the site logo, navigation links, and a search bar. Below the header is a large advertisement for Volkswagen TDI Clean Diesel Event, featuring a red car and promotional text. The main content area contains several articles, including one titled "In New York, Hard Choices on Health" and another titled "An Artist Takes His Pay in Coffee and Community". A social media widget for @NYTMETRO ON TWITTER is also visible. The page is annotated with three callout boxes: "Page code" points to the browser's address bar and the top navigation area; "Ad code" points to the Volkswagen advertisement; and "Third-party APIs" points to the Twitter widget.

Modern web "site"

Page code

Ad code

Third-party libraries

Third-party APIs

The screenshot shows a browser window displaying the New York Times website. The page features a navigation bar with the site logo, search, and regional options. A prominent advertisement for Volkswagen TDI Clean Diesel is visible, offering a lease for \$289/month. Below the ad, there are several content blocks: a news article titled "In New York, Hard Choices on Health" with a photo of a doctor's office; a "SIDE STREET" column about an artist; a "BIG CITY BOOK CLUB" section; a "VIDEO" player featuring a man; and a "FASHION" article titled "Intersection: Elmhurst Style Ease". A Twitter widget for @NYTMETRO is also present. Red boxes and lines connect the callout text on the left to these specific elements on the page.

Modern web "site"

Page code

Ad code

Extensions

Third-party libraries

Third-party APIs

The screenshot shows a browser window displaying a page from 'The New York Times' with the URL 'www.nytimes.com/pages/nyregion...'. The page features a navigation bar with 'The New York Times' logo, 'N.Y. / Region', and buttons for 'SUBSCRIBE NOW', 'SIGN IN', and 'Register'. Below the navigation bar is a large advertisement for Volkswagen TDI Clean Diesel Event, featuring a red car and the text 'Lease a 2014 Golf TDI for \$289/mo for 36 months. \$0 Down'. The main content area includes a headline 'In New York, Hard Choices on Health' with a sub-headline 'An Artist Takes His Pay in Coffee and Community' and a video player. A sidebar on the right contains a Twitter widget for '@NYTMETRO ON TWITTER' and a video player for 'Intersection: Elmhurst Style Ease'. Red boxes highlight the browser's address bar, the Volkswagen advertisement, the Twitter widget, and the video player. Callout lines connect these highlighted areas to text boxes on the left: 'Page code' points to the browser's address bar; 'Ad code' points to the Volkswagen advertisement; 'Extensions' points to the browser's toolbar; 'Third-party libraries' points to the Twitter widget; and 'Third-party APIs' points to the video player.

Sites handle sensitive data

- **Financial data**
 - Online banking, tax filing, shopping, budgeting, ...
- **Health data**
 - Genomics, prescriptions, ...
- **Personal data**
 - Email, messaging, affiliations, ...

Others want this information

- **Financial data**
 - Black-hat hackers, ...
- **Health data**
 - Insurance companies, ...
- **Personal data**
 - Ad companies, big governments, ...

Others want this information

- **Financial data**
 - Black-hat hackers, ...
- **Health data**
 - Insurance companies, ...
- **Personal data**
 - Ad companies, big governments, ...



Others want this information

- Financial data
 - Black-hat hackers, ...
- Health data
 - Insurance companies, ...
- Personal data
 - Ad companies, big governments, ...



Others want this information

- Financial data
 - Black-hat hackers, ...
- Health data
 - Insurance companies, ...
- Personal data
 - Ad companies, big governments, ...



The acting parties on a site

- Page developer
- Library developers
- Service providers
- Data provides
- Ad providers
- Other users
- CDNs
- Extension developers

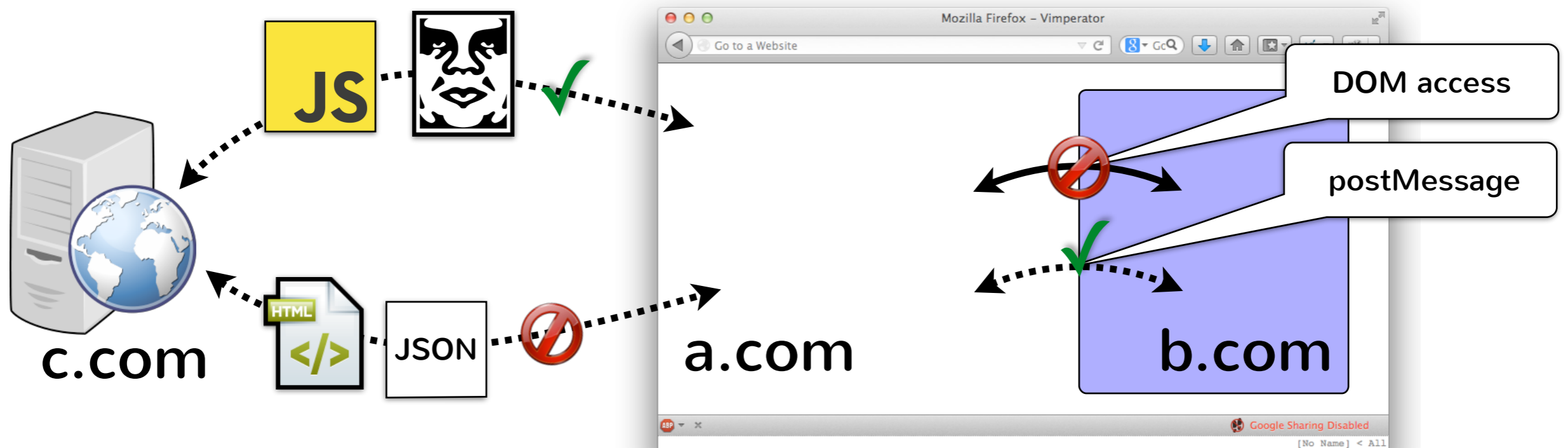
Basic questions

- How do we protect page from ads/services?
- How to share data with cross-origin page?
- How to protect one user from another's content?
- How do we protect the page from a library?
- How do we protect page from CDN?
- How do we protect extension from page?

Recall: Same origin policy

Idea: isolate content from different origins

- E.g., can't access document of cross-origin page
- E.g., can't inspect responses from cross-origin



**Is the same origin policy good
enough?**

The SOP is not strict enough

- Third-party libs run with privilege of the page
- Code within page can arbitrarily leak data
 - How?
- iframes isolation is limited
 - Can't isolate user-provided content from page (why?)
 - Can't isolate third-party ad placed in iframe (why?)

The SOP is not strict enough

- Third-party libs run with privilege of the page
- Code within page can arbitrarily leak data
 - How?
- iframes isolation is limited

- Can't isolate us
- Can't isolate th



The SOP is not flexible enough

- Can't read cross-origin responses
 - What if we want to fetch data from provider.com?
 - JSONP
 - To fetch data, insert new script tag:
`<script src="https://provider.com/getData?cb=f"></script>`
 - To share data, reply back with script wrapping data
`f({ ...data... })`
 - Why is this a terrible idea?
 - Provider data can easily be leaked (CSRF)
 - Page is not protected from provider (XSS)

The SOP is not flexible enough

- Can't read cross-origin responses
 - What if we want to fetch data from provider.com?
 - JSONP
 - To fetch data, insert new script tag:
`<script src="https://provider.com/getData?cb=f"></script>`
 - To share data, reply back with script wrapping data
`f({ ...data... })`
 - Why is this a terrible idea?
 - Provider data can easily be leaked (CSRF)
 - Page is not protected from provider (XSS)

Outline: modern mechanisms

- **iframe sandbox**
- **Content security policy (CSP)**
- **Web workers**
 - Not originally intended for security; but they help
- **Subresource integrity (SRI)**
- **Cross-origin resource sharing (CORS)**

iframe sandbox

Idea: restrict actions iframe can perform

Approach: set sandbox attribute, by default:

- disallows JavaScript and triggers (autofocus, autoplay videos etc.)
- disallows form submission
- disallows popups
- disallows navigating embedding page
- runs page in unique origin: no storage/cookies

Whitelisting privileges

Can enable dangerous features by whitelisting:

- **allow-scripts:** allows JS + triggers (autofocus, autoplay, etc.)
- **allow-forms:** allow form submission
- **allow-pointer-lock:** allow fine-grained mouse moves
- **allow-popups:** allow iframe to create popups
- **allow-top-navigation:** allow breaking out of frame
- **allow-same-origin:** retain original origin

What can you do with iframe sandbox?

- **Run content in iframe with least privilege**
 - Only grant content privileges it needs
- **Privilege separate page into multiple iframes**
 - Split different parts of page into sandboxed iframes

E.g., least privilege: twitter button

```
<a class="twitter-share-button" href="https://twitter.com/share">Tweet</a>
<script>
window.twttr=(function(d,s,id){var js,fjs=d.getElementsByTagName(s)
[0],t=window.twttr||{};if(d.getElementById(id))return
t;js=d.createElement(s);js.id=id;js.src="https://platform.twitter.com/
widgets.js";fjs.parentNode.insertBefore(js,fjs);t._e=[];t.ready=function(f)
{t._e.push(f);};return t;}(document,"script","twitter-wjs"));
</script>
```

- What's the problem with this embedding approach?

- **Using iframes**

```
<iframe src="https://platform.twitter.com/widgets/tweet_button.html"
style="border: 0; width:130px; height:20px;"></iframe>
```

- What's the problem with this approach?

E.g., least privilege: twitter button

- With sandbox: remove all permissions and then enable JS, popups, form submission, etc.

```
<iframe src="https://platform.twitter.com/widgets/tweet_button.html"  
  sandbox="allow-same-origin allow-scripts allow-popups allow-forms"  
  style="border: 0; width:130px; height:20px;"></iframe>
```

- Why is are these required (e.g., same origin)?

E.g., privilege separation: feed

- Typically include user content inline:

```
<div class="post">  
  <div class="author">{{post.author}}</div>  
  <div class="body">{{post.body}}</div>  
</div>
```

- Problem with this?

- With iframe sandbox:

```
<iframe sandbox srcdoc="...  
<div class="post">  
  <div class="author">{{post.author}}</div>  
  <div class="body">{{post.body}}</div>  
</div>..."></iframe>
```

- May need allow-scripts - why? allow-same-origin ok?

Basic questions

- ✓ How do we protect page from ads/services?
 - How to share data with cross-origin page?
- ✓ How to protect one user from another's content?
 - How do we protect the page from a library?
 - How do we protect page from CDN?
 - How do we protect extension from page?

Limitations/questions on sandbox

- **Research: How can you determine what privileges you need to run a page with?**
 - Being overly restricting: breaks functionality
 - Being overly permissive: can cause more damage
- **Research: Automatically compartmentalization?**
- **Is the loose definition of "least privilege" good enough?**
 - It mostly restricts features, not what you can do with the features; what can go wrong?

Motivation for CSP

- Consider running library in sandboxed iframes
 - E.g., password strength checker



- **Desired guarantee:** checker cannot leak password
- Problem: sandbox does not restrict exfiltration
 - Can use XHR to write password to b.ru

Motivation for CSP

- Can we limit the origins that the page (iframe or otherwise) can talk to?
 - Can only leak to a trusted set of origins
 - Gives us a more fine-grained notion of least privilege
- Can we extend this idea to prevent or limit damages due to XSS?

Content security policy

- Goal: prevent or limit damage due to XSS
- Idea: restrict resource loading to a white list
 - By restricting to whom page can talk to: restrict where data is leaked!
- Approach: send page with CSP header that contains fine-grained directives
 - E.g., allow loads from CDN, no frames, no plugins

```
Content-Security-Policy: default-src https://cdn.example.net;  
child-src 'none'; object-src 'none'
```

Example directives

- **connect-src**: limits the origins you can XHR to
- **font-src**: where to fetch web fonts from
- **form-action**: where forms can be submitted
- **child-src**: where to load frames/workers from
- **frame-ancestors**: sources that can embed this page
- **default-src**: default whitelist

Special keywords

- 'none' - match nothing
- 'self' - match this origin
- 'unsafe-inline' - allow unsafe JS & CSS
- 'unsafe-eval' - allow unsafe eval (and the like)
- http: - match anything with http scheme
- https: - match anything with https scheme
- * - match anything

How can CSP prevent XSS?

- **If you whitelist all places you can load scripts from:**
 - Only execute code from trusted origins
 - Remaining vector for attack: inline scripts
- **CSP by default disallows inline scripts**
 - If scripts are enabled at least it disallows eval

Adoption challenge

- **Problem: inline scripts are widely-used**
 - Page authors use the 'unsafe-inline' directive
 - Is this a problem?
- **Solution: script nonce and script hash**
 - Allow scripts that have a particular hash
 - Allow scripts that have a white-listed nonce

Other adoption challenges

- Goal: set most restricting CSP that is permissive enough to not break existing app
- How can you figure this out for a large app?
- CSP has report-only header and report-uri directive
 - Report violations to server; don't enforce

Basic questions

- ✓ How do we protect page from ads/services?
 - How to share data with cross-origin page?
- ✓ How to protect one user from another's content?
- ✓ How do we protect the page from a library?
 - How do we protect page from CDN?
 - How do we protect extension from page?

Limitations/questions on CSP

- Can still exfiltrate data (postMessage, navigation)
- Research: setting flexible CSP policy automatically
 - Dynamic loading content vs. CSP (Reddit imgurl)
- Research: set CSP automatically with inline scripts in presence of user-supplied content?
 - Stored XSS problem: user code vs. your inline code
- Research [COWL]: is whitelisting enough?

Web workers

- **Run code in separate context (in new thread)**
 - No DOM: no postMessage to iframes/navigation to leak
 - Only pure JavaScript + XHR + postMessage/onmessage with parent
- **CSP header on worker can be more restricting than page**
 - A more secure sandbox for running untrusted code

Outline: modern mechanisms

- **iframe sandbox**
- **Content security policy (CSP)**
- **Web workers**
 - Not originally intended for security; but they help
- ➔ **Subresource integrity (SRI)**
- **Cross-origin resource sharing (CORS)**

Motivation for SRI

- CSP can be used to limit the damage of code, but can't really defend against malicious code
- How do you know that the library you're loading is the correct one?

Massive denial-of-service attack on GitHub tied to Chinese government

Reports: Millions of innocent Internet users conscripted into Chinese DDoS army.

- Won't using HTTPS address this problem?

Motivation for SRI

- CSP can be used to limit the damage of code, but can't really defend against malicious code
- How do you know that the library you're loading is the correct one?

Massive denial-of-service attack on GitHub tied to Chinese government

Reports: Millions of innocent Internet users conscripted into Chinese DDoS army.

- Won't using HTTPS address this problem?

jQuery.com compromised to serve malware via drive-by download

Subresource integrity

- Idea: page author specifies hash of (sub)resource they are loading; browser checks integrity

- E.g., integrity for scripts

```
<link rel="stylesheet" href="https://site53.cdn.net/style.css" integrity="sha256-SDfwewFAE...wefjijfE">
```

- E.g., integrity for link elements

```
<script src="https://code.jquery.com/jquery-1.10.2.min.js" integrity="sha256-C6CB9UYIS9UJeqinPHWTHVqh/E1uhG5Tw+Y5qFQmYg=">
```

What happens when check fails?

- **Case 1 (default):**
 - Browser reports violation and does not render/execute resource
- **Case 2: CSP directive with integrity-policy directive set to report**
 - Browser reports violation, but may render/execute resource

Multiple hash algorithms

- Authors may specify multiple hashes

- E.g.,

```
<script src="hello_world.js"
      integrity="sha256-...
               sha512-...
"></script>
```

- Browser uses strongest algorithm

- Why support multiple algorithms?

- Don't break page on old browser

Multiple hash algorithms

- Authors may specify multiple hashes
 - E.g.,

```
<script src="hello_world.js"
      integrity="sha256-...
               sha512-...
"></script>
```
- Browser uses strongest algorithm
- Why support multiple algorithms?
 - Don't break page on old browser

Basic questions

- ✓ How do we protect page from ads/services?
 - How to share data with cross-origin page?
- ✓ How to protect one user from another's content?
- ✓ How do we protect the page from a library?
- ✓ How do we protect page from CDN?
 - How do we protect extension from page?

Limitations/questions on SRI

- Only supports stylesheets and scripts
- Can extend to other elements? UI integrity?
- Can extend to downloads?
- Research: what if you used signatures?
 - Talk to Henry Corrigan-Gibbs and Amit Levy

Outline: modern mechanisms

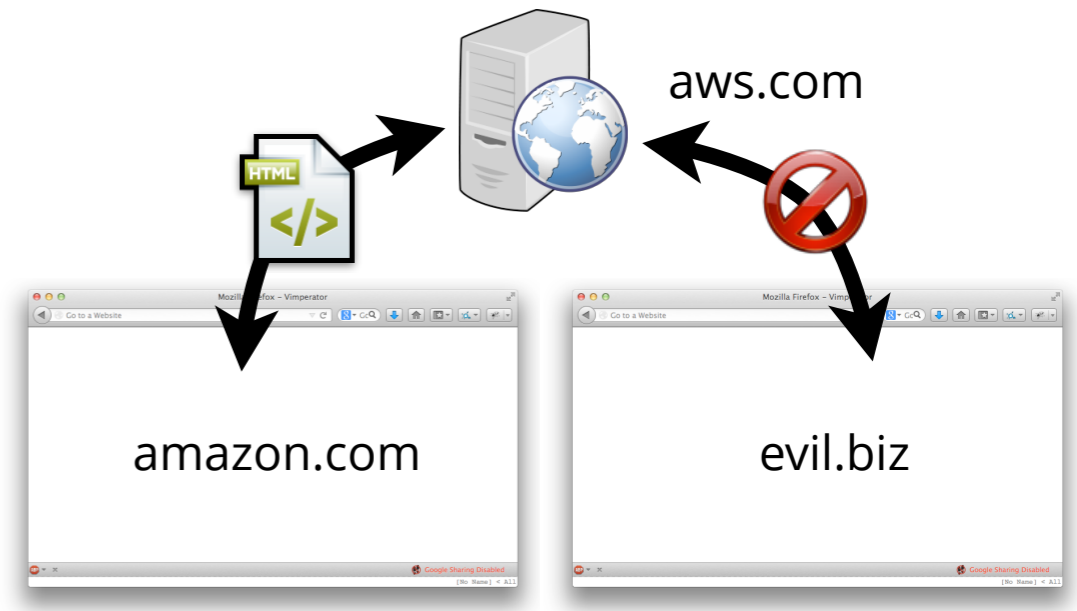
- **iframe sandbox**
- **Content security policy (CSP)**
- **Web workers**
 - Not originally intended for security; but they help
- **Subresource integrity (SRI)**
- ➔ **Cross-origin resource sharing (CORS)**

Recall: SOP is also inflexible

- **Problem: Can't fetch cross-origin data**
 - Leads to building insecure sites/services: JSONP
- **Solution: Cross-origin resource sharing (CORS)**
 - Data provider explicitly whitelists origins that can inspect responses
 - Browser allows page to inspect response if its origin is listed in the header

E.g., CORS usage: amazon

- Amazon has multiple domains
 - E.g., amazon.com and aws.com
- Problem: amazon.com can't read cross-origin aws.com data
- With CORS amazon.com can whitelist aws.com



How CORS works

- **Browser sends Origin header with XHR request**
 - E.g., Origin: `https://amazon.com`
- **Server can inspect Origin header and respond with Access-Control-Allow-Origin header**
 - E.g., Access-Control-Allow-Origin: `https://amazon.com`
 - E.g., Access-Control-Allow-Origin: `*`
- **CORS XHR may send cookies + custom headers**
 - Need “preflight” request to authorize this

Basic questions

- ✓ How do we protect page from ads/services?
- ✓ How to share data with cross-origin page?
- ✓ How to protect one user from another's content?
- ✓ How do we protect the page from a library?
- ✓ How do we protect page from CDN?
- How do we protect extension from page?

Limitations/questions on CORS

- Can't share data with sandboxed iframe without making it completely public
- Research [COWL]: is whitelisting enough?
 - Why doesn't chase.com share bank statements with mint.com?
- Research: CORS + crypto for better sharing?

Outline: modern mechanisms

- **iframe sandbox**
- **Content security policy (CSP)**
- **Web workers**
 - Not originally intended for security; but they help
- **Subresource integrity (SRI)**
- **Cross-origin resource sharing (CORS)**

How do we protect extensions from pages?

- **Firefox and Chrome:**
 - Isolated worlds: extension script's heap is different from the heap of the page. Why?
 - E.g., `getElementById = function() {...evil stuff...}`

How do we protect extensions from pages?

- **Chrome forces developers to follow:**
 - Privilege separation by breaking extension into
 - Core extension script: has access to privileged APIs
 - Content script: can manipulate page but must ask core script to use privileged APIs on its behalf
 - Principle of least privileged via permission system
 - User must approve APIs granted to core extension scripts, so developers should be kept in line

Basic questions

- ✓ How do we protect page from ads/services?
- ✓ How to share data with cross-origin page?
- ✓ How to protect one user from another's content?
- ✓ How do we protect the page from a library?
- ✓ How do we protect page from CDN?
- ✓ How do we protect extension from page?

Limitations/questions on extension systems

- **Page can't protect itself from extension**
 - Extensions do directly inject code and have removed CSP headers [RAID]
- **Research [HotOS]: is trust model realistic? Is Chrome's system working? Can we do better?**
 - Extensions are third-party code; there have been malicious extensions in the wild
 - Extensions are not least privileged: over 71% of top 1000 need to read/write everything for every origin

Continuing w/ research questions

- Can we build an extension systems with more realistic attacker model?
- Where do existing mechanisms for the Web fall short?

Motivation for COWL (working spec draft)

- Same Origin Policy
- Content Security Policy
- Sandboxing



Motivation for COWL (working spec draft)

- Same Origin Policy
- Content Security Policy
- Sandboxing



All-or-nothing discretionary access control:
access data **⇒** ability to leak it

Where DAC falls short...

Where DAC falls short...

Third-party APIs

New password: Password strength: **Strong**

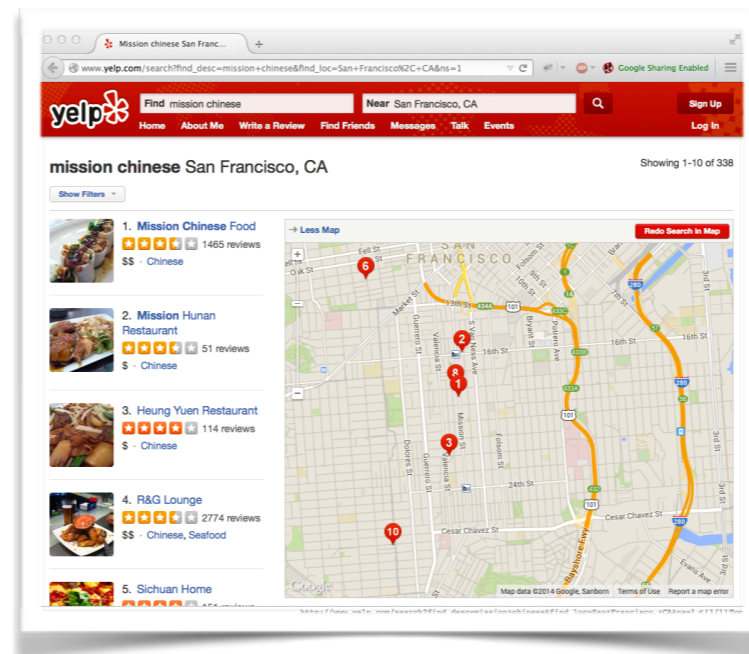
A screenshot of a password strength indicator. It shows a label 'New password:' followed by a password input field with 12 black dots. To the right is a label 'Password strength:' with a blue underline, followed by the word 'Strong' in green. Below the 'Password strength:' label is a green horizontal bar.

Where DAC falls short...

Third-party APIs

New password: Password strength: **Strong**

Mashups

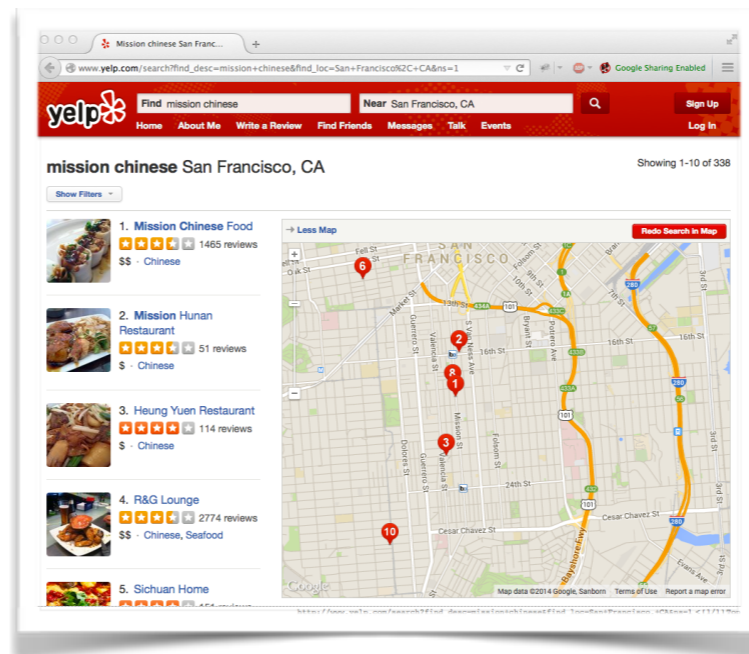


Where DAC falls short...

Third-party APIs



Mashups



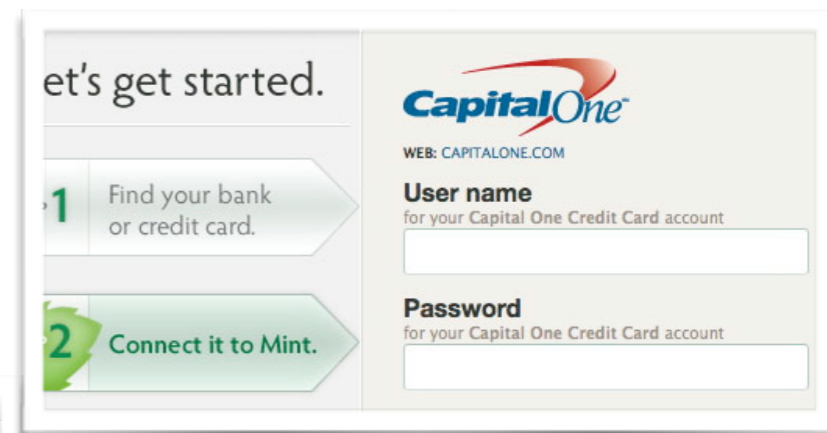
Third-party libraries

Where DAC falls short...

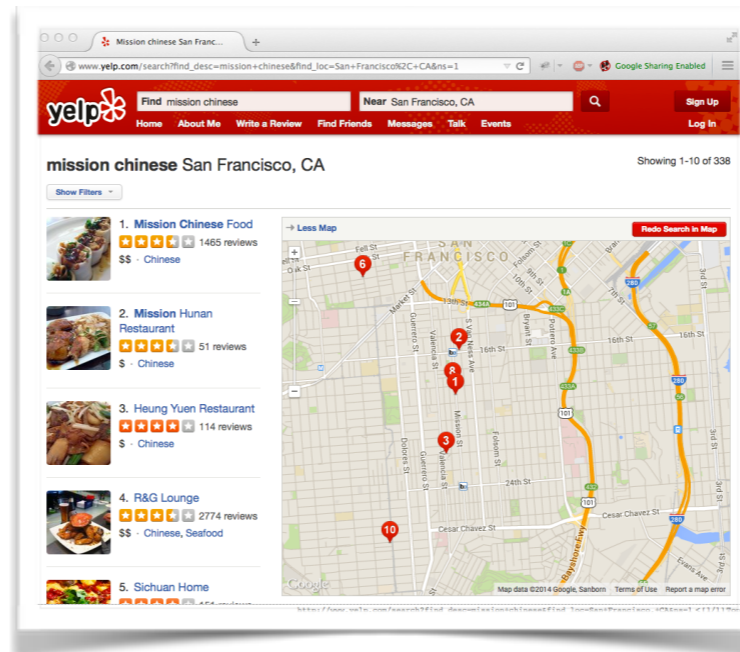
Third-party APIs



Third-party mashups



Mashups



Third-party libraries



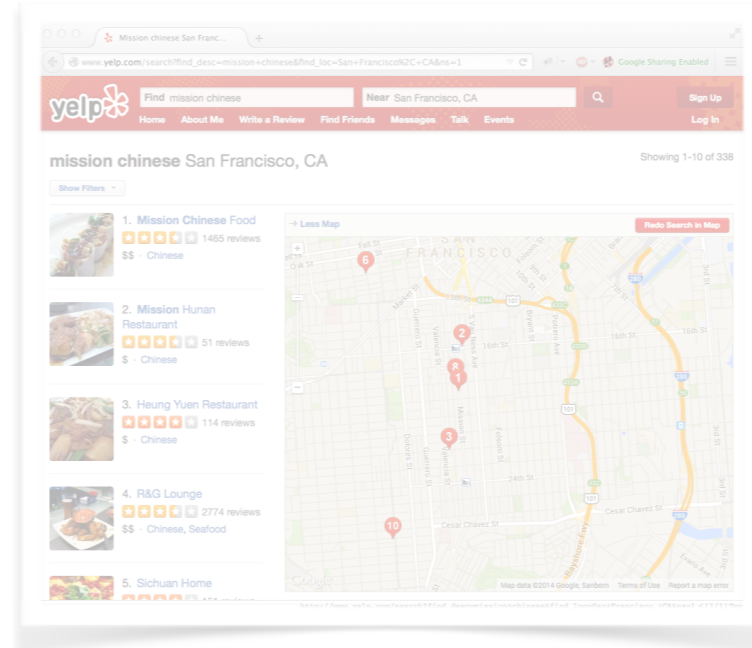
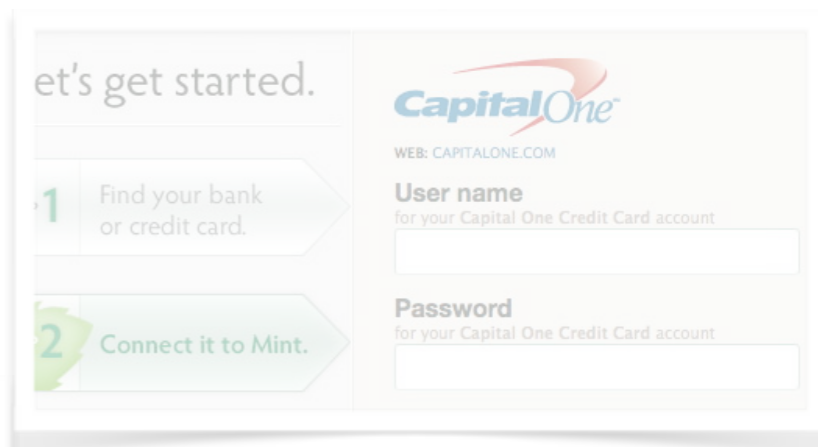
Where DAC falls short...

Third-party APIs



Third-party mashups

Mashups



Third-party libraries

Recall: password-strength checker

New password:

Password strength: **Strong**

a.com

b.ru/chk.html

Guarantee: checker cannot leak password

- At worst: checker lies about strength of password

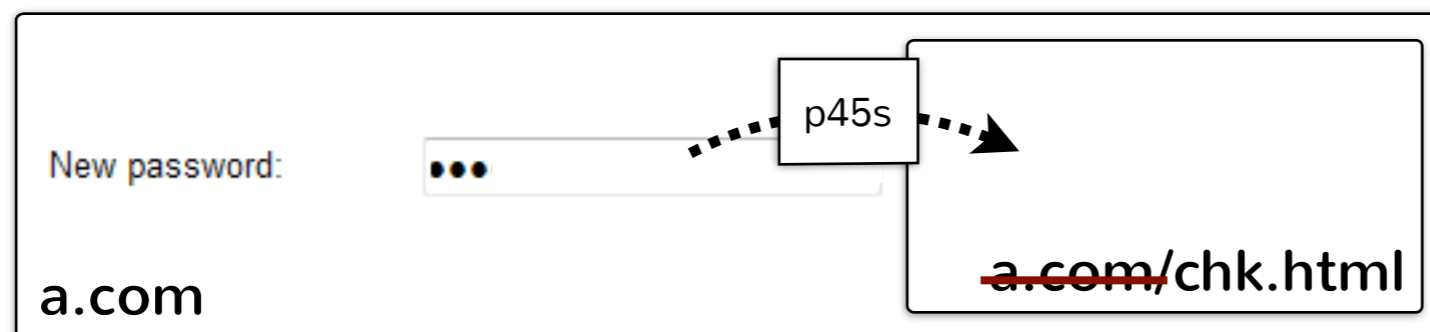
Confining the checker using existing mechanisms

- Host the checker code on a.com
- Use CSP & Sandboxing
 - Need JavaScript: `sandbox allow-scripts`
 - Limit communication to postMessage with parent: `default-src 'none' 'unsafe-inline'`



Confining the checker using existing mechanisms

- Host the checker code on a.com
- Use CSP & Sandboxing
 - Need JavaScript: `sandbox allow-scripts`
 - Limit communication to postMessage with parent: `default-src 'none' 'unsafe-inline'`



Confining the checker using existing mechanisms

- Host the checker code on a.com
- Use CSP & Sandboxing
 - Need JavaScript: `sandbox allow-scripts`
 - Limit communication to postMessage with parent: `default-src 'none' 'unsafe-inline'`



Confining the checker using existing mechanisms

- Host the checker code on a.com
- Use CSP & Sandboxing
 - Need JavaScript: `sandbox allow-scripts`
 - Limit communication to postMessage with parent: `default-src 'none' 'unsafe-inline'`



Confining the checker using existing mechanisms

- Host the checker code on a.com
- Use CSP & Sandboxing
 - Need JavaScript: `sandbox allow-scripts`
 - Limit communication to `postMessage` with parent: `default-src 'none' 'unsafe-inline'`



Confining the checker using existing mechanisms

- Host the checker code on a.com
- Use CSP & Sandboxing
 - Need JavaScript: `sandbox allow-scripts`
 - Limit communication to postMessage with parent: `default-src 'none' 'unsafe-inline'`



Confining the checker using existing mechanisms

- Host the checker code on a.com
- Use CSP & Sandboxing
 - Need JavaScript: `sandbox allow-scripts`
 - Limit communication to `postMessage` with parent: `default-src 'none' 'unsafe-inline'`



Confining the checker using existing mechanisms

- Host the checker code on a.com
- Use CSP & Sandboxing
 - Need JavaScript: `sandbox allow-scripts`
 - Limit communication to `postMessage` with parent: `default-src 'none' 'unsafe-inline'`



Confining the checker using existing mechanisms

- Host the checker code on a.com
- Use CSP & Sandboxing
 - Need JavaScript: `sandbox allow-scripts`
 - Limit communication to `postMessage` with parent:
`default-src 'none' 'unsafe-inline'`

Actually can leak to iframes, so need to use Worker...



Why is this unsatisfactory?

- **Functionality of library is limited**
 - E.g., library cannot fetch resources from network
 - A more flexible CSP policy would weaken security
- **Security policy is not first-class**
 - Library cannot use code it itself doesn't trust
- **Security policy is not symmetric**
 - Library cannot consider parent untrusted

A new approach: COWL

Idea (a): Provide means for associating security label with data

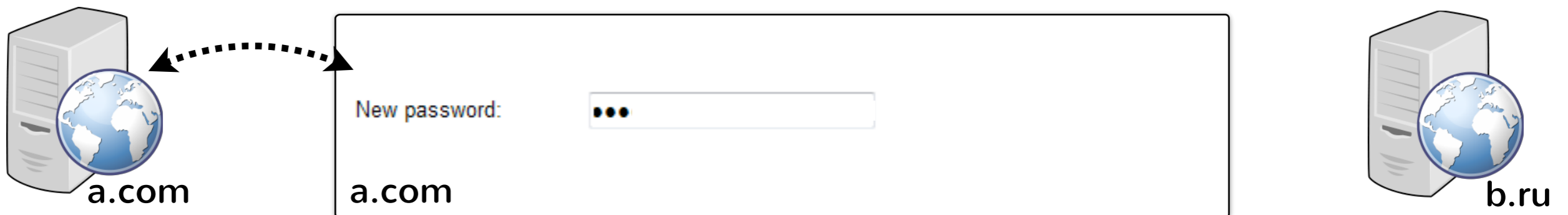
- E.g., password is sensitive to a.com

Idea (b): Ensure code is confined to obey labels by associating labels with browsing contexts

- E.g., password can only be sent to entities that are as sensitive as a.com
(via XHR, postMessage, storage, ...)

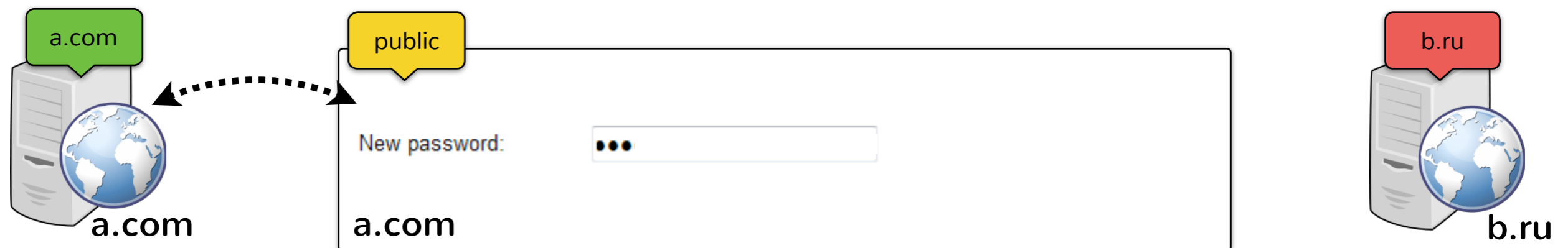
Confining the checker with COWL

- Express sensitivity of data
 - Checker can only receive password if its context label is as sensitive as the password
- Use `postMessage` to send labeled password
 - Source specifies sensitivity of data at time of send



Confining the checker with COWL

- Express sensitivity of data
 - Checker can only receive password if its context label is as sensitive as the password
- Use `postMessage` to send labeled password
 - Source specifies sensitivity of data at time of send



Confining the checker with COWL

- Express sensitivity of data
 - Checker can only receive password if its context label is as sensitive as the password
- Use `postMessage` to send labeled password
 - Source specifies sensitivity of data at time of send



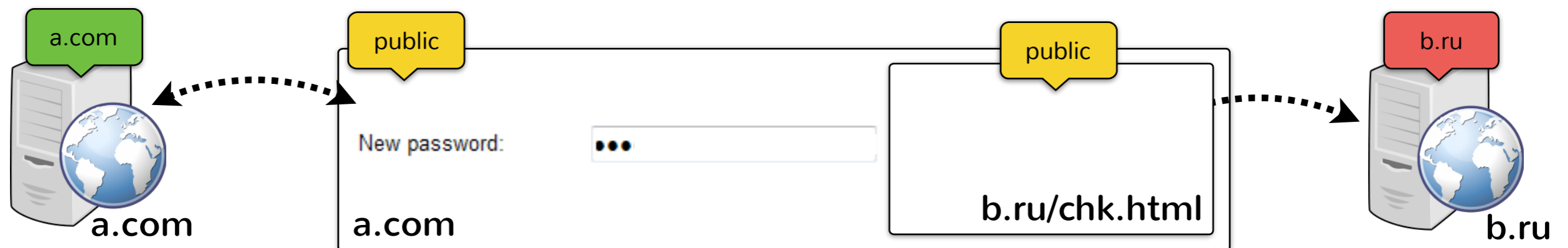
Confining the checker with COWL

- Express sensitivity of data
 - Checker can only receive password if its context label is as sensitive as the password
- Use `postMessage` to send labeled password
 - Source specifies sensitivity of data at time of send



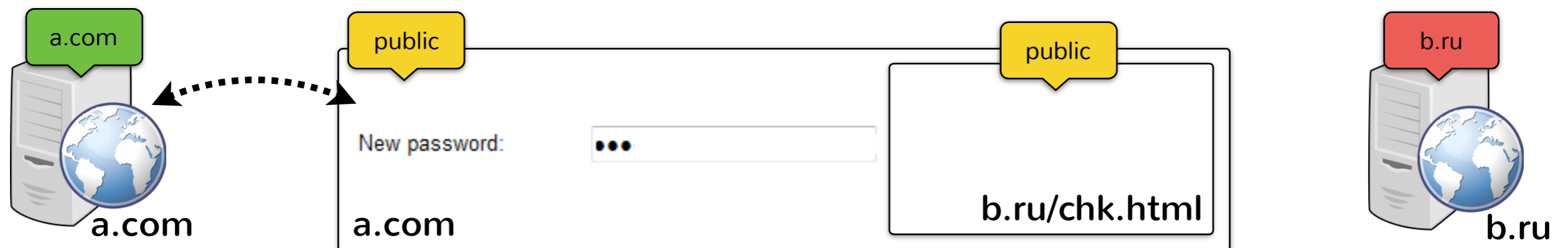
Confining the checker with COWL

- Express sensitivity of data
 - Checker can only receive password if its context label is as sensitive as the password
- Use `postMessage` to send labeled password
 - Source specifies sensitivity of data at time of send



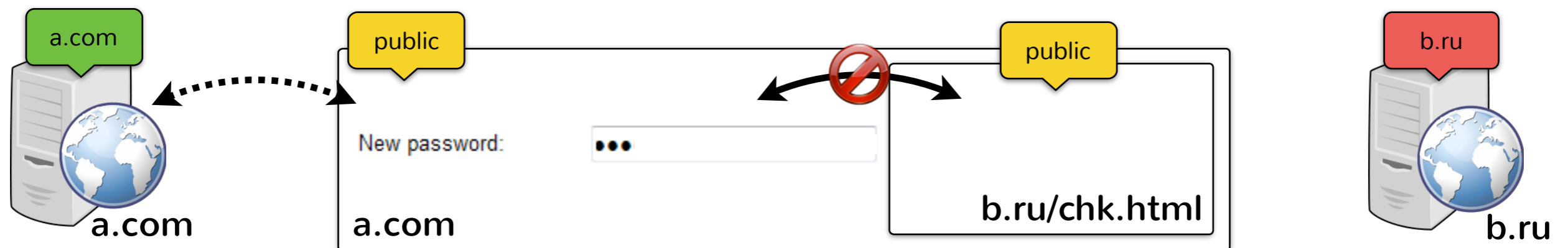
Confining the checker with COWL

- Express sensitivity of data
 - Checker can only receive password if its context label is as sensitive as the password
- Use `postMessage` to send labeled password
 - Source specifies sensitivity of data at time of send



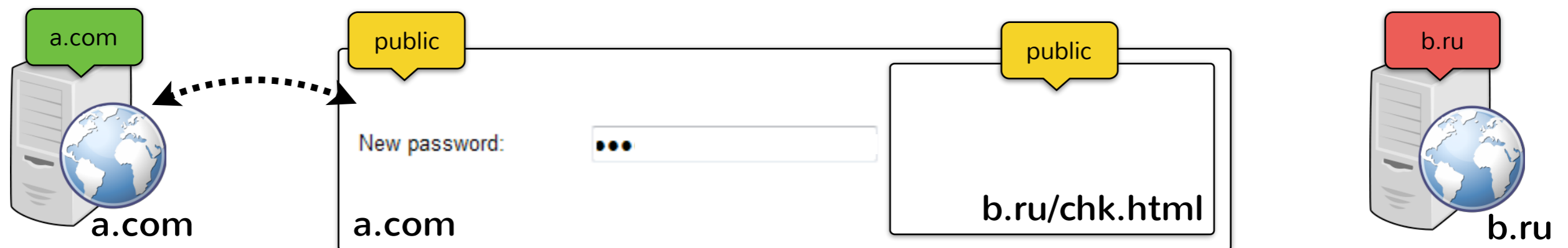
Confining the checker with COWL

- Express sensitivity of data
 - Checker can only receive password if its context label is as sensitive as the password
- Use `postMessage` to send labeled password
 - Source specifies sensitivity of data at time of send



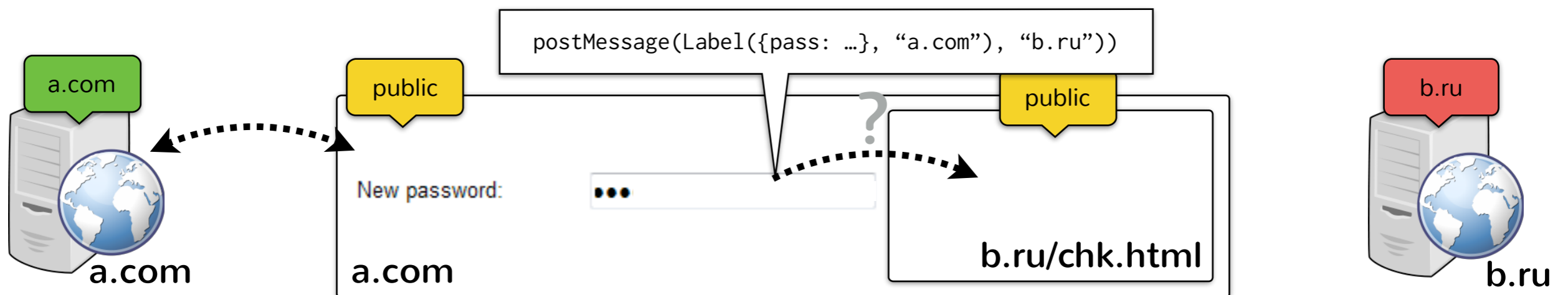
Confining the checker with COWL

- Express sensitivity of data
 - Checker can only receive password if its context label is as sensitive as the password
- Use postMessage to send labeled password
 - Source specifies sensitivity of data at time of send



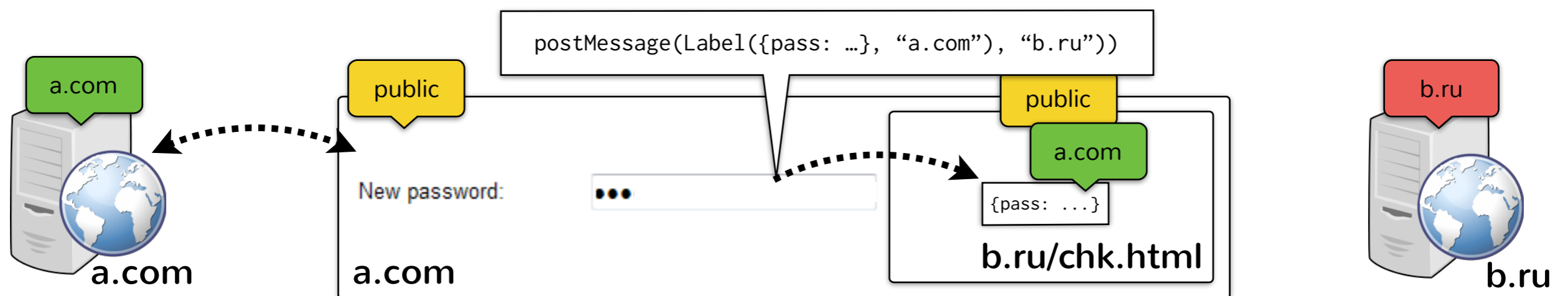
Confining the checker with COWL

- Express sensitivity of data
 - Checker can only receive password if its context label is as sensitive as the password
- Use `postMessage` to send labeled password
 - Source specifies sensitivity of data at time of send



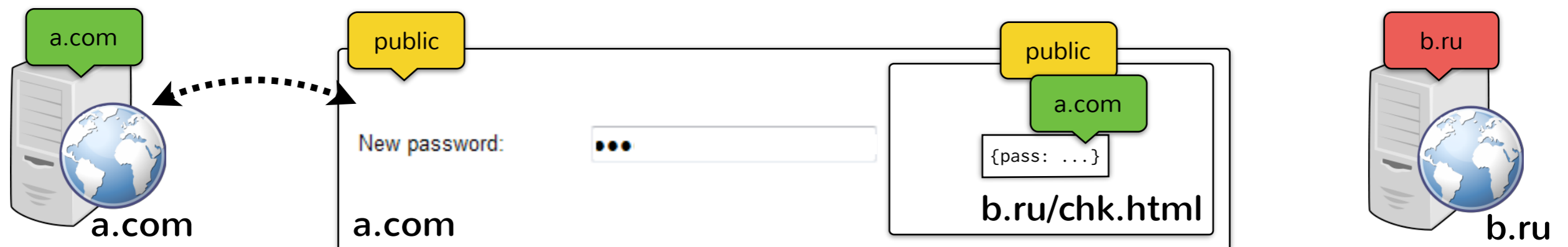
Confining the checker with COWL

- Express sensitivity of data
 - Checker can only receive password if its context label is as sensitive as the password
- Use `postMessage` to send labeled password
 - Source specifies sensitivity of data at time of send



Confining the checker with COWL

- Express sensitivity of data
 - Checker can only receive password if its context label is as sensitive as the password
- Use postMessage to send labeled password
 - Source specifies sensitivity of data at time of send



Confining the checker with COWL

- Express sensitivity of data
 - Checker can only receive password if its context label is as sensitive as the password
- Use postMessage to send labeled password

- Source specifies sensitivity

```
onmessage = function (labeledPass) {  
  var pass = unlabel(labeledPass);  
  var strength = checkStrength(pass);  
  ...  
}
```

send



Confining the checker with COWL

- Express sensitivity of data
 - Checker can only receive password if its context label is as sensitive as the password
- Use postMessage to send labeled password

- Source specifies sensitivity

```
onmessage = function (labeledPass) {  
  var pass = unlabel(labeledPass);  
  var strength = checkStrength(pass);  
  ...  
}
```

send



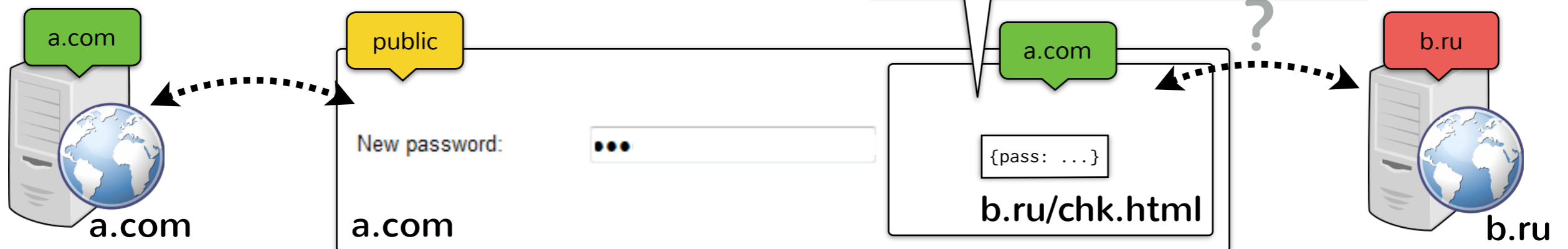
Confining the checker with COWL

- Express sensitivity of data
 - Checker can only receive password if its context label is as sensitive as the password
- Use `postMessage` to send labeled password

- Source specifies sensitivity

```
onmessage = function (labeledPass) {  
  var pass = unlabel(labeledPass);  
  var strength = checkStrength(pass);  
  ...  
}
```

send



Confining the checker with COWL

- Express sensitivity of data
 - Checker can only receive password if its context label is as sensitive as the password
- Use postMessage to send labeled password

- Source specifies sensitivity

```
onmessage = function (labeledPass) {  
  var pass = unlabel(labeledPass);  
  var strength = checkStrength(pass);  
  ...  
}
```

send



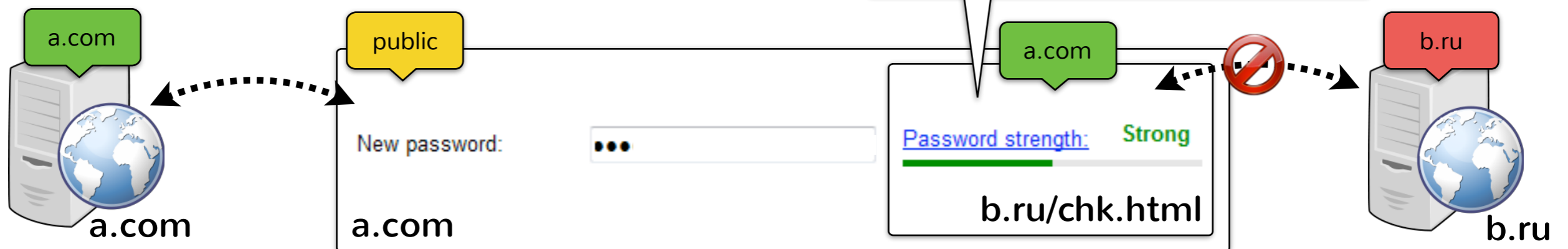
Confining the checker with COWL

- Express sensitivity of data
 - Checker can only receive password if its context label is as sensitive as the password
- Use postMessage to send labeled password

- Source specifies sensitivity

```
onmessage = function (labeledPass) {  
  var pass = unlabel(labeledPass);  
  var strength = checkStrength(pass);  
  ...  
}
```

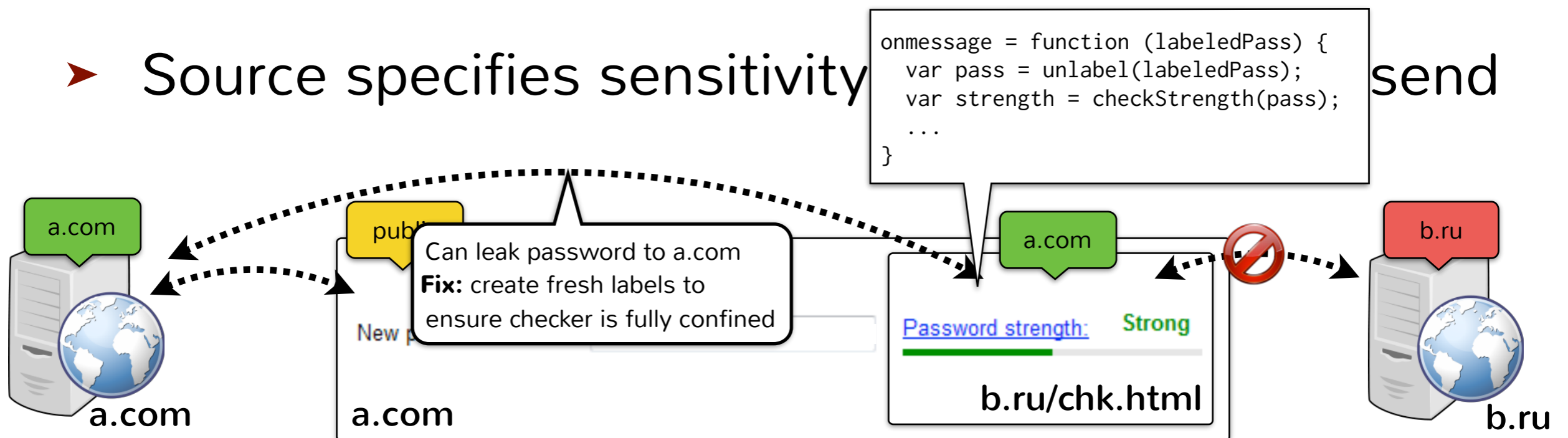
send



Confining the checker with COWL

- Express sensitivity of data
 - Checker can only receive password if its context label is as sensitive as the password
- Use postMessage to send labeled password

- Source specifies sensitivity



Summary

- **SOP has reached its limit for modern web apps**
- **New mechanisms: sandboxing, CSP, CORS, SRI**
 - Address limitations of SOP by reducing amount of trust authors need to place in code (by reducing the amount of damage code can cause)
 - Each has their own shortcomings
 - COWL address limitation of whitelists
 - Signatures can address limitations of SRI
 - Lot of work to do
- **Web apps do not run stand-alone: extensions**
 - Extension systems protect privileged code from untrusted app code, though design needs revising

References

- [Sandbox] - **Play safely in sandboxed IFrames** by Mike West.
 - <http://www.html5rocks.com/en/tutorials/security/sandboxed-iframes/>
 - <http://www.w3.org/TR/2010/WD-html5-20100624/the-iframe-element.html>
- [CSP] - **An Introduction to Content Security Policy** by Mike West.
 - <http://www.html5rocks.com/en/tutorials/security/content-security-policy/>
 - <http://www.w3.org/TR/CSP2/>
- [CORS] - **Using CORS** by Monsur Hossain.
 - <http://www.html5rocks.com/en/tutorials/cors/>
 - <http://www.w3.org/TR/cors>
- [SRI] - **Subresource Integrity** by Frederik Braun, Francois Marier, Devdatta Akhawe, and Joel Weinberger.
 - <http://www.w3.org/TR/SRI>

References

- [COWL] - **Protecting Users by Confining JavaScript with COWL** by Deian Stefan, Edward Z. Yang, Petr Marchenko, Alejandro Russo, Dave Herman, Brad Karp, and David Mazières. In OSDI 2014
 - <http://cowl.ws>
- [HotOS] - **The Most Dangerous Code in the Browser** by Stefan Heule, Devon Rifkin, Deian Stefan, and Alejandro Russo. In HotOS 2015
 - <http://deian.org/pubs/heule:2015:the-most.pdf>
- [RAID] - **Why is CSP Failing? Trends and Challenges in CSP Adoption** by Michael Weissbacher, Tobias Lauinger, and William Robertson. In RAID, 2014.
 - <http://www.iseclab.org/people/mweissbacher/publications/csp RAID.pdf>