The Web's Security Model

Philippe De Ryck – iMinds-DistriNet, KU Leuven

philippe.deryck@cs.kuleuven.be

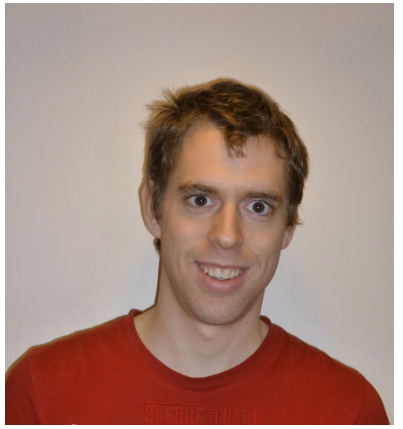
DistriNet iMinds KU LEUVEN

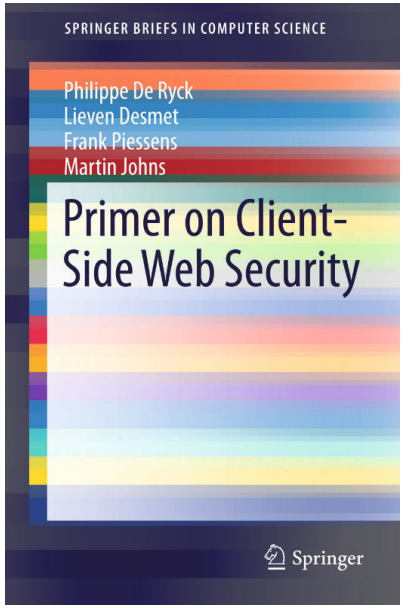
STREWS

SecureApplication Development
secappdev.org

About Me – Philippe De Ryck

- Postdoctoral Researcher @ DistriNet (KU Leuven)
  - Focus on (client-side) Web security
- Responsible for the Web Security training program
  - Dissemination of knowledge and research results
  - Target audiences include industry and researchers
- Main author of the *Primer on Client-Side Web Security*
  - 7 attacker models, broken down in 10 capabilities
  - 13 attacks and their countermeasures
  - Overview of security best practices





DistriNet iMinds KU LEUVEN STREWS

Introducing *example.com*

Public Information

Analytics

Private Customer Area

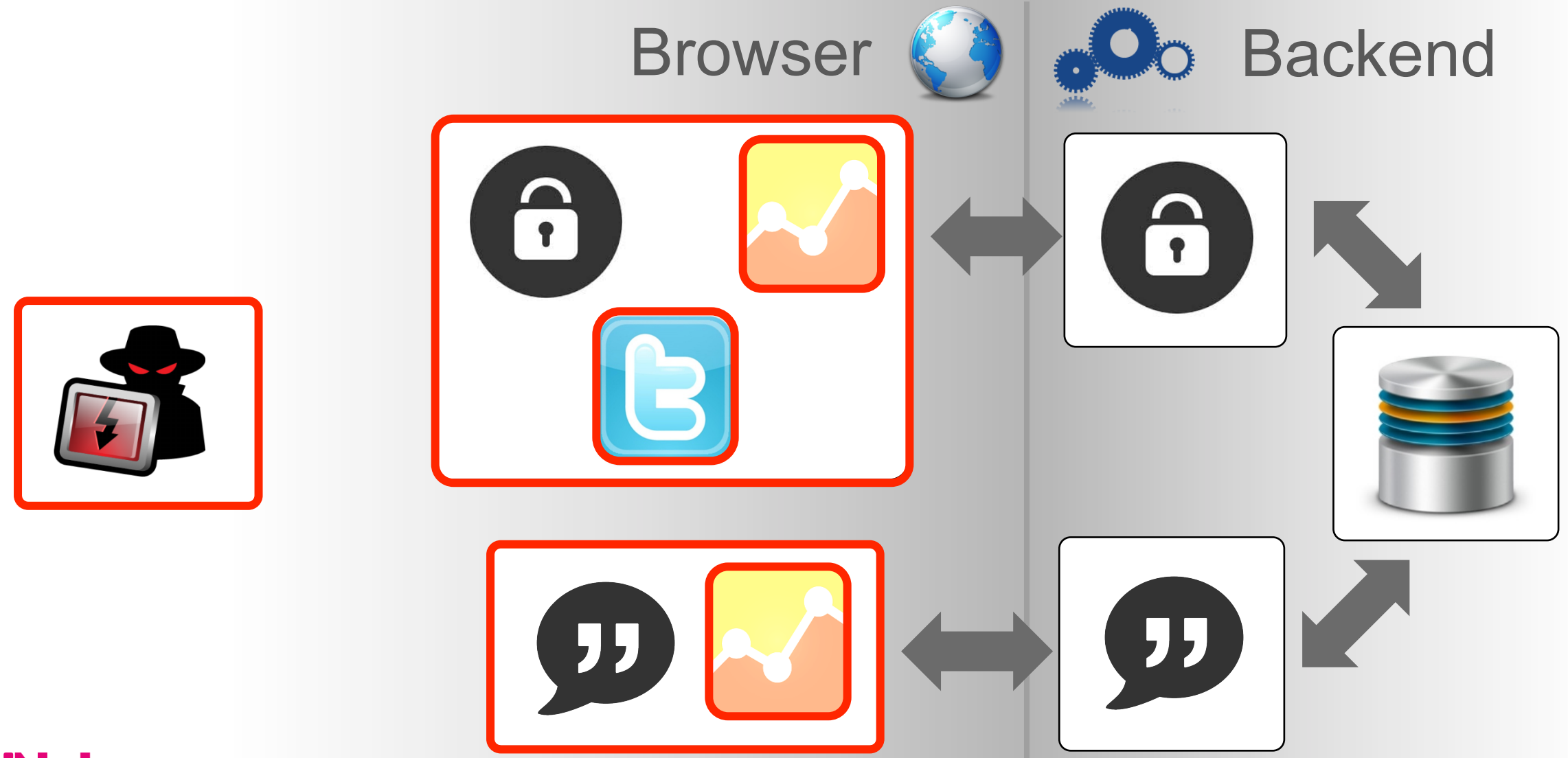
Twitter Integration

Account Management

Location Information

Public Forum

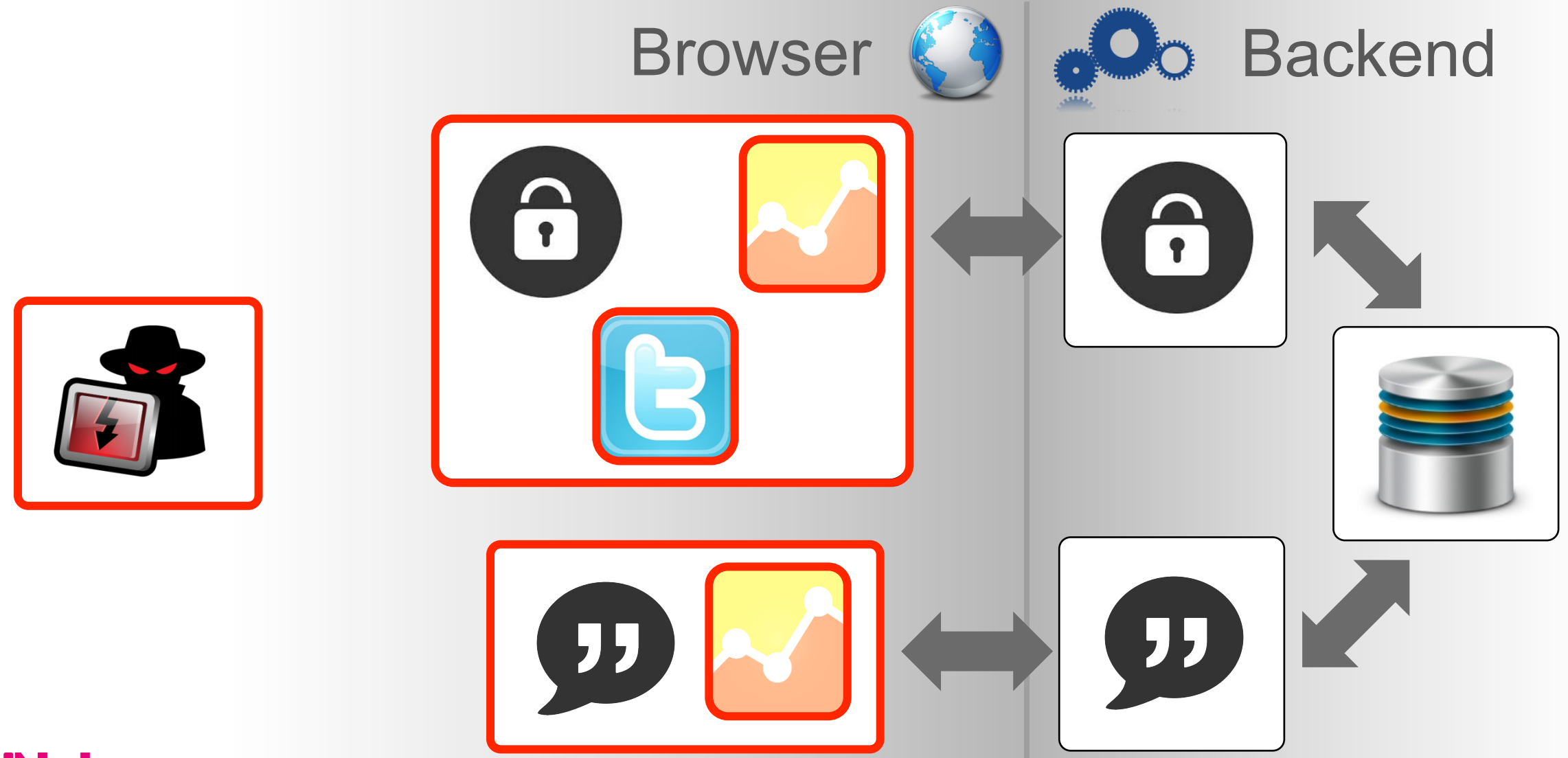
Deploying *example.com*



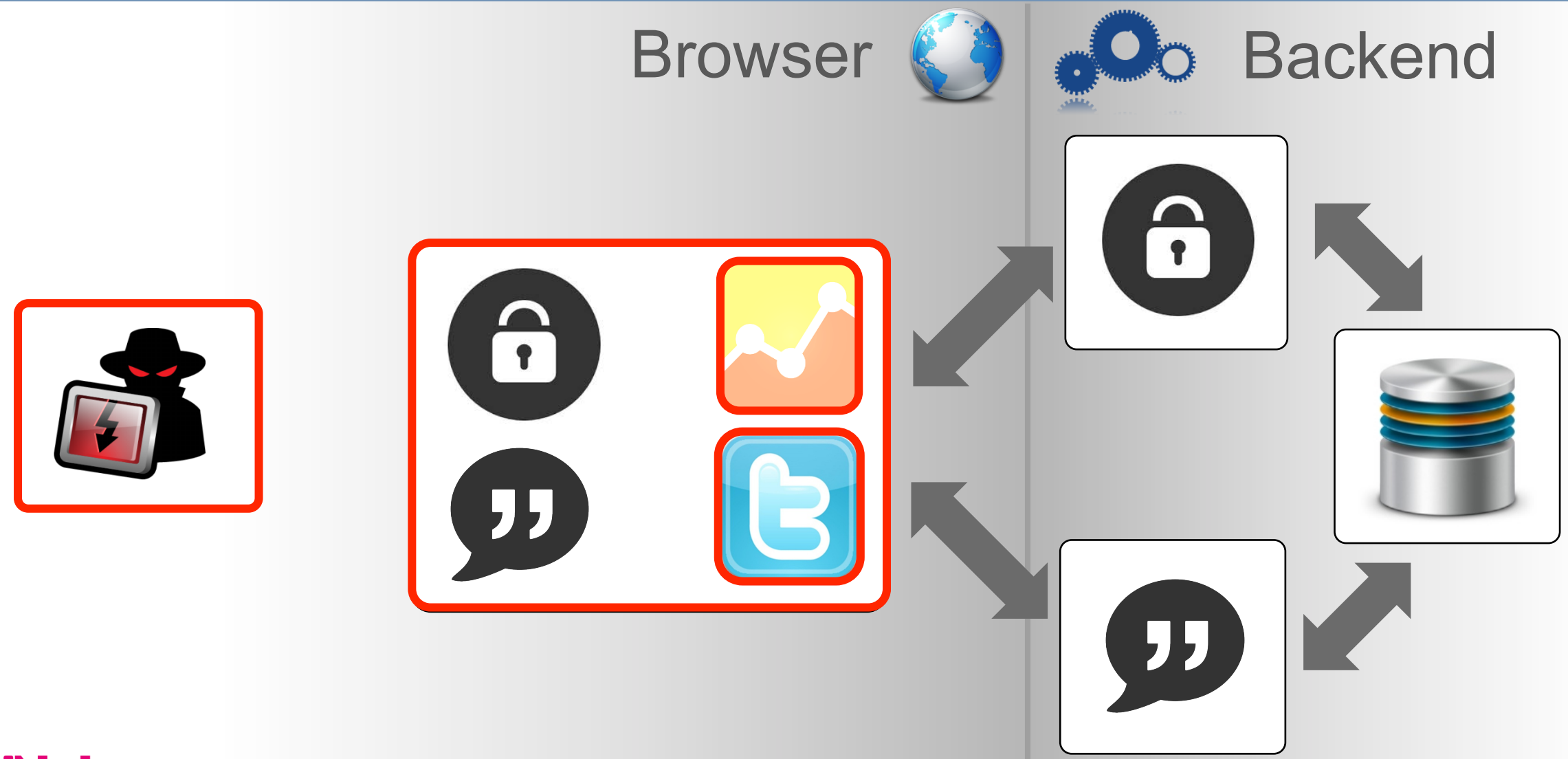
Deploying *example.com* in the Web

It can’t be that simple, right?

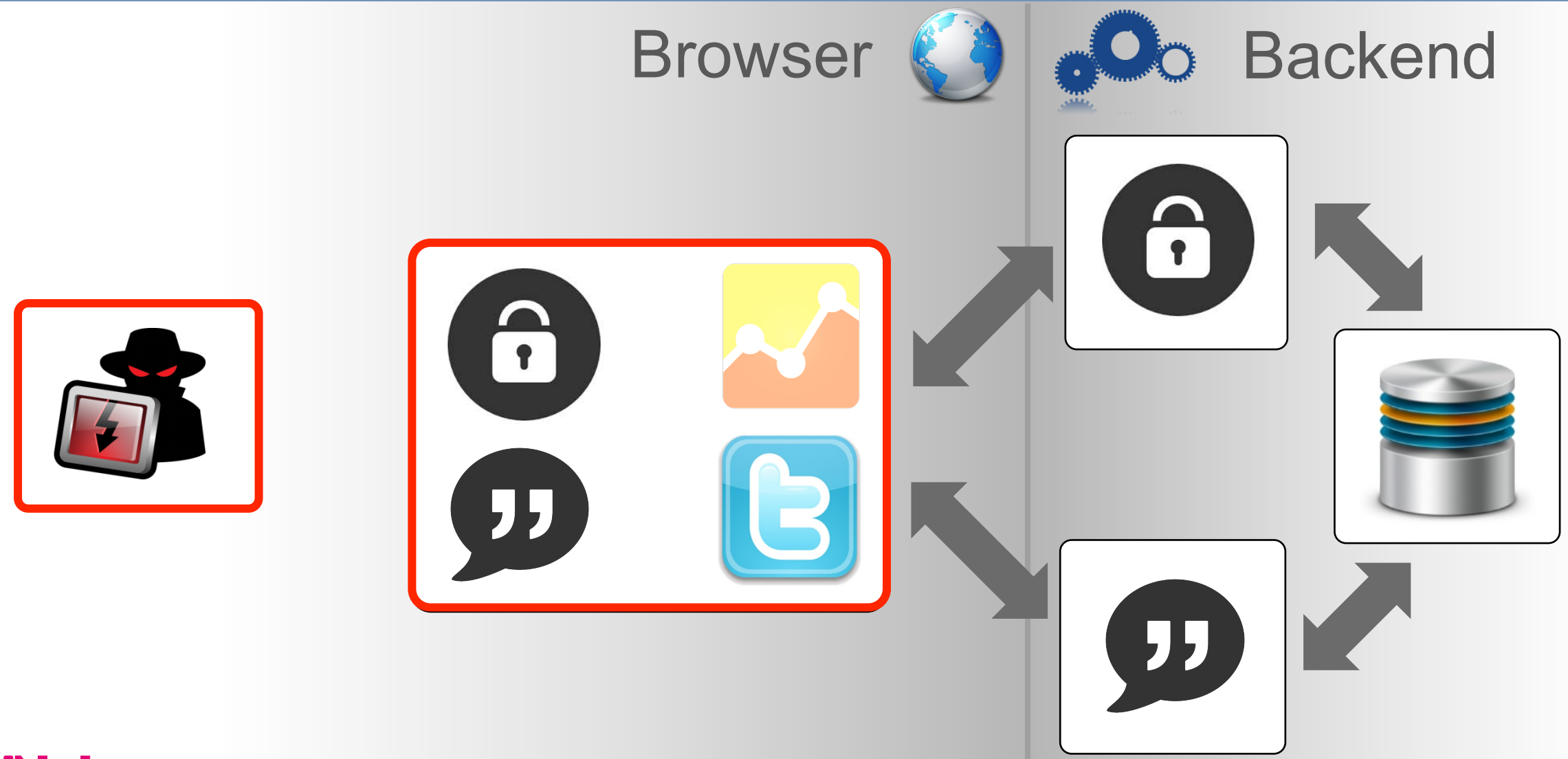
Deploying *example.com*



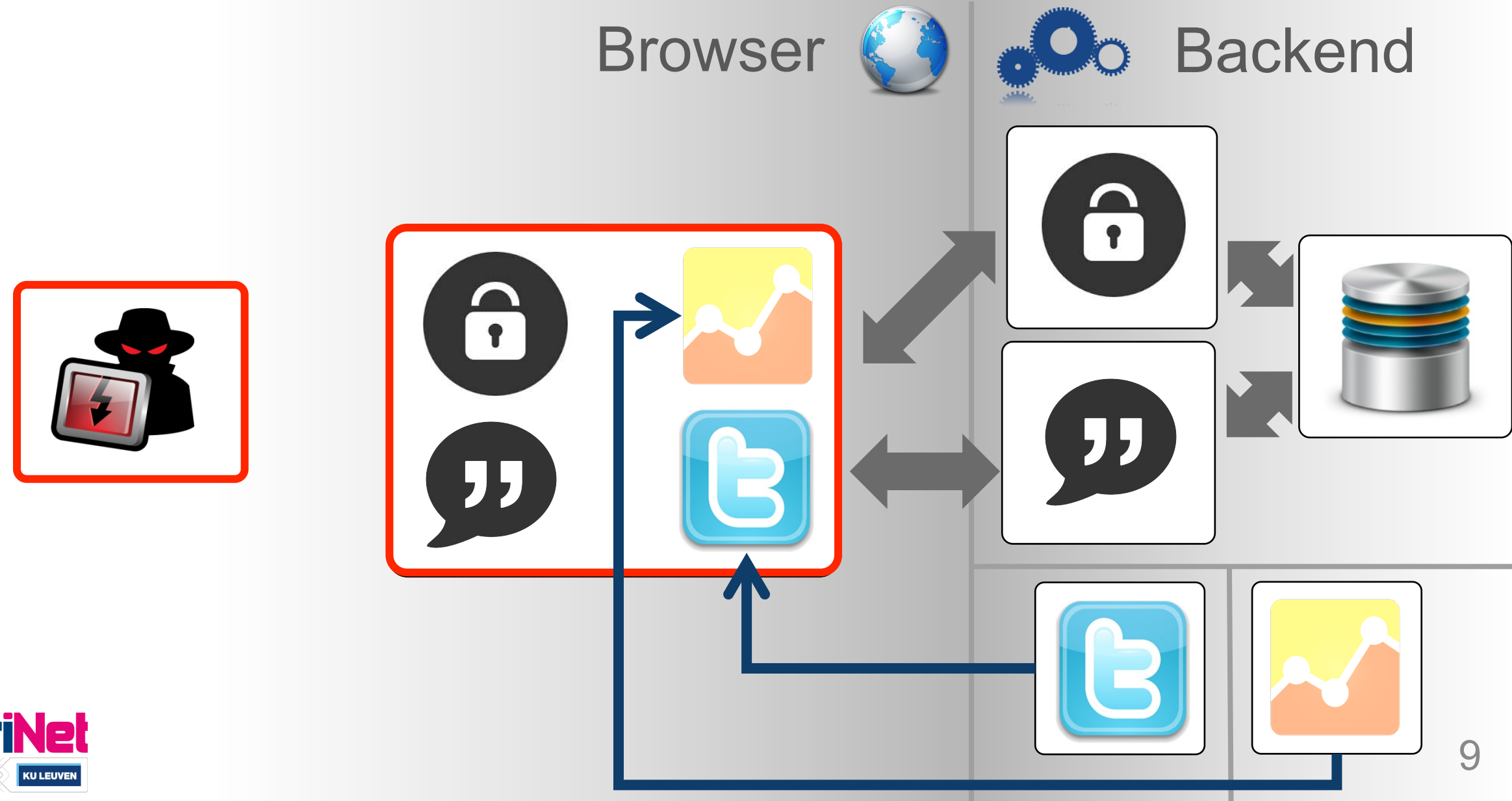
Origin-based Isolation



Integration of Third-party Components



Remote Inclusion of Third-party Scripts

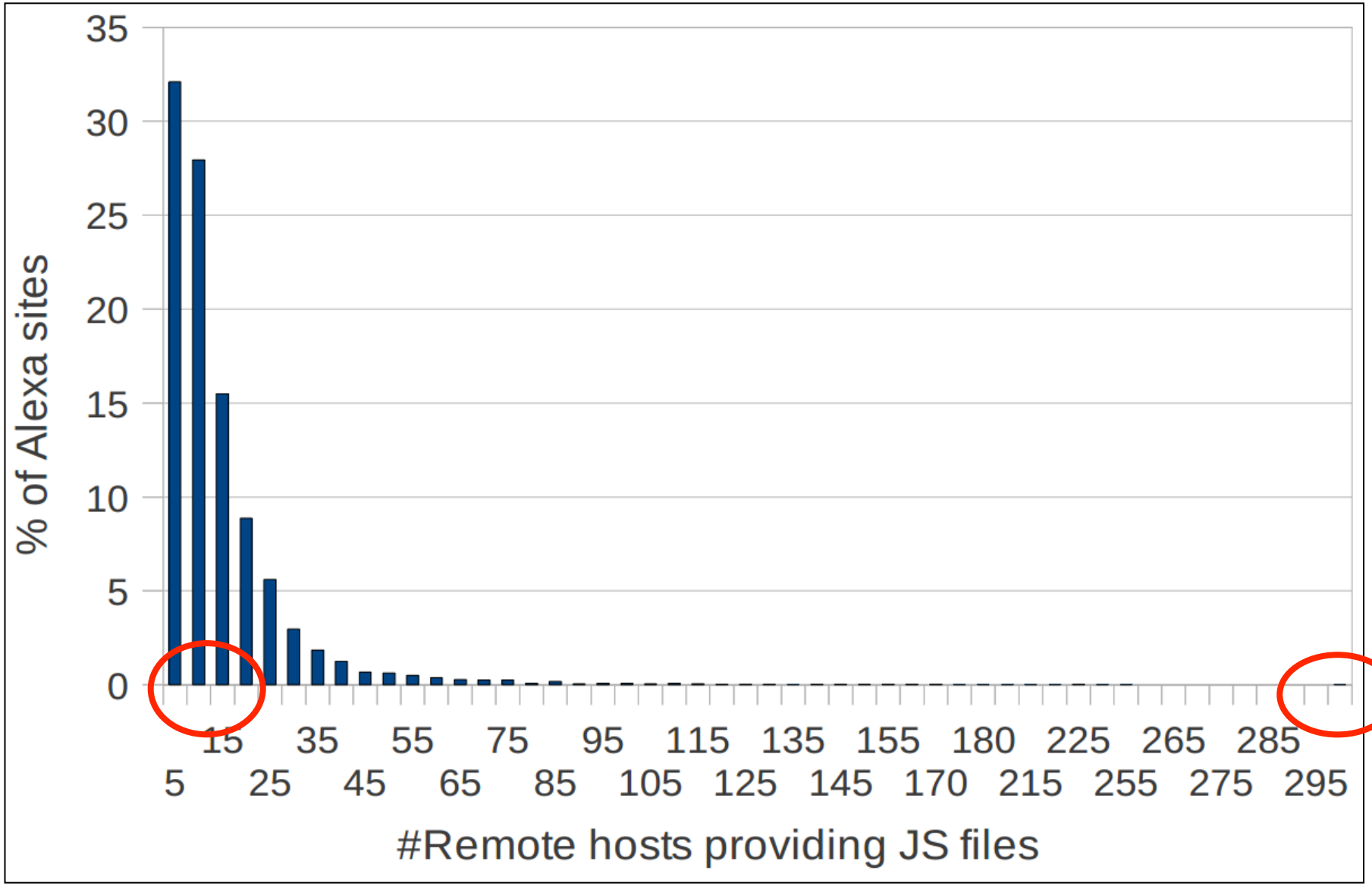


Compromise of Third-party Providers

If you downloaded the qTip2 library between *8th December 2011 and 10th of January 2012*, please make sure to re-download the library as the site was compromised between these dates due to malicious code injected via a Wordpress bug. Apologies for any inconvinience caused by this, but as usual vulnerabilities like this can only be pro-actively remedied as they occur.

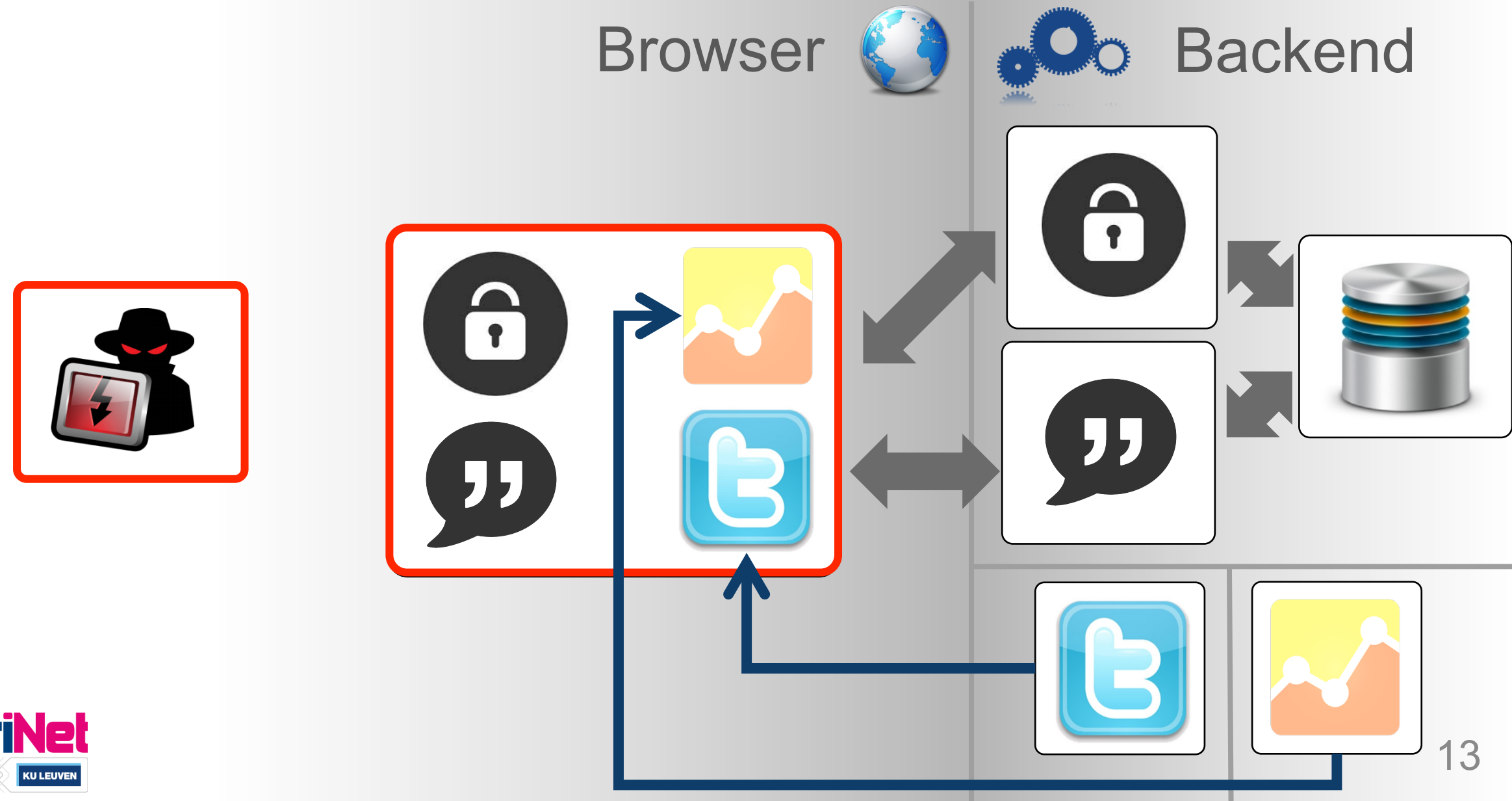
Large-scale Study of Remote JS Inclusions

Large-scale Study of Remote JS Inclusions

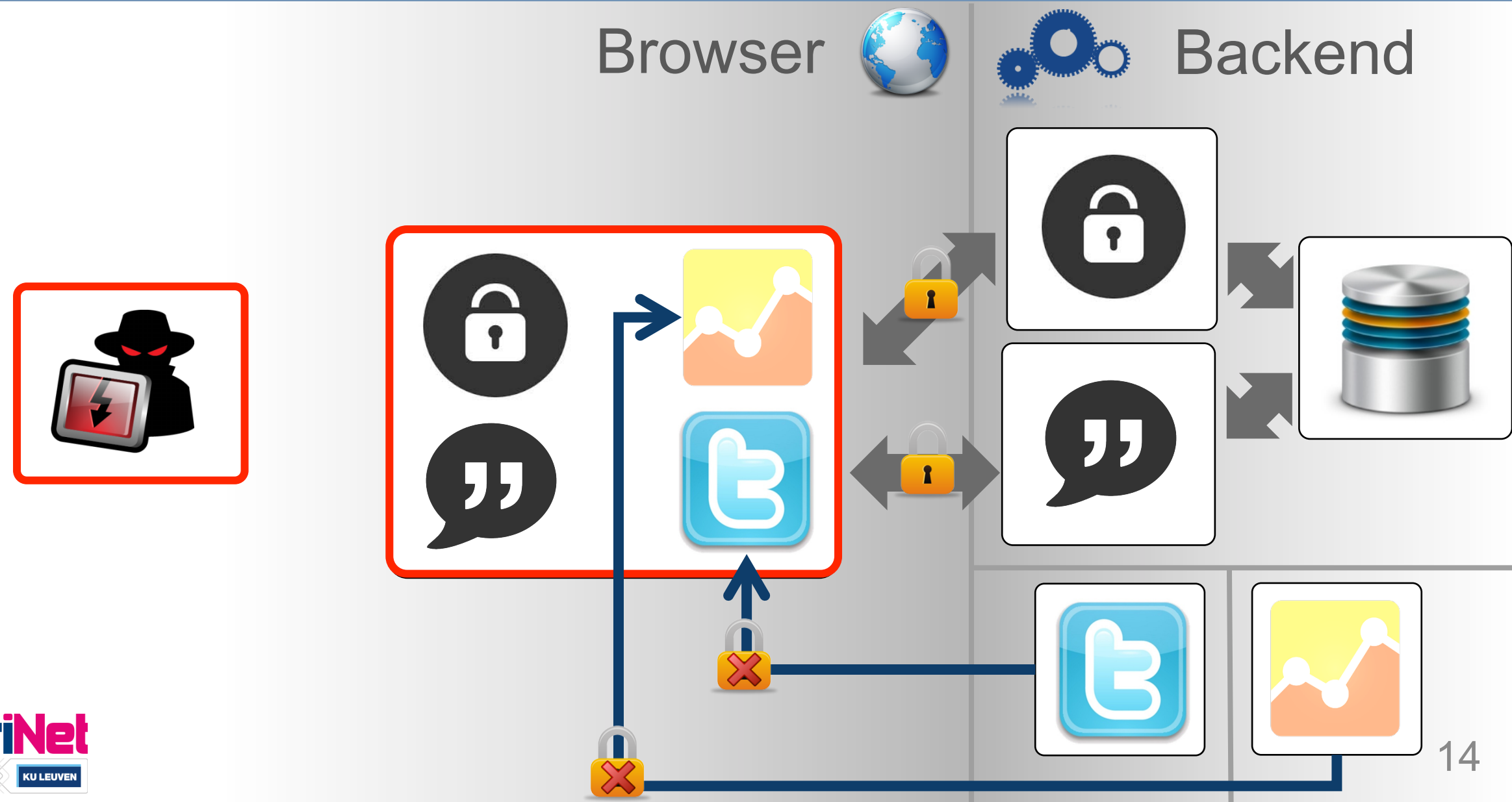


DistriNet
iMinds
KU LEUVEN

Remote Inclusion of Third-party Scripts

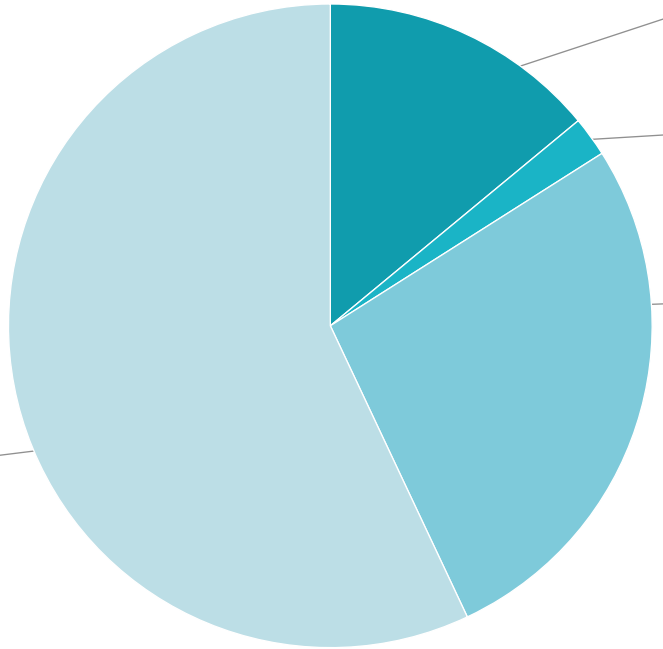


Mixed Content Inclusions



Large-scale Study of Mixed Content

“43% of 18,526 HTTPS sites in the Alexa top 100,000 has at least one mixed content inclusion”



14% exposed to request forgery and cookie stealing

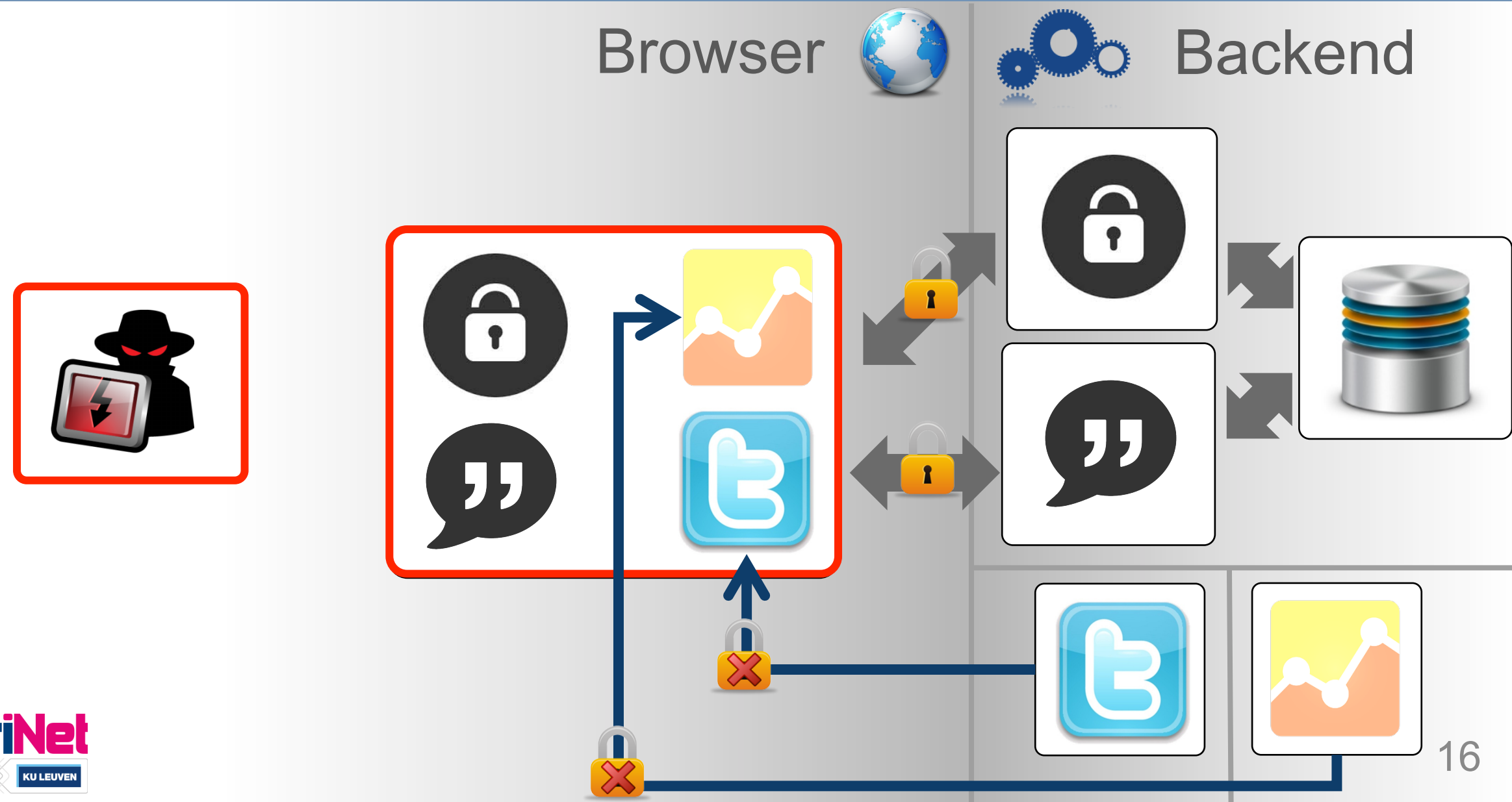
2% exposed to DOM data leakage

27% exposed to JavaScript execution

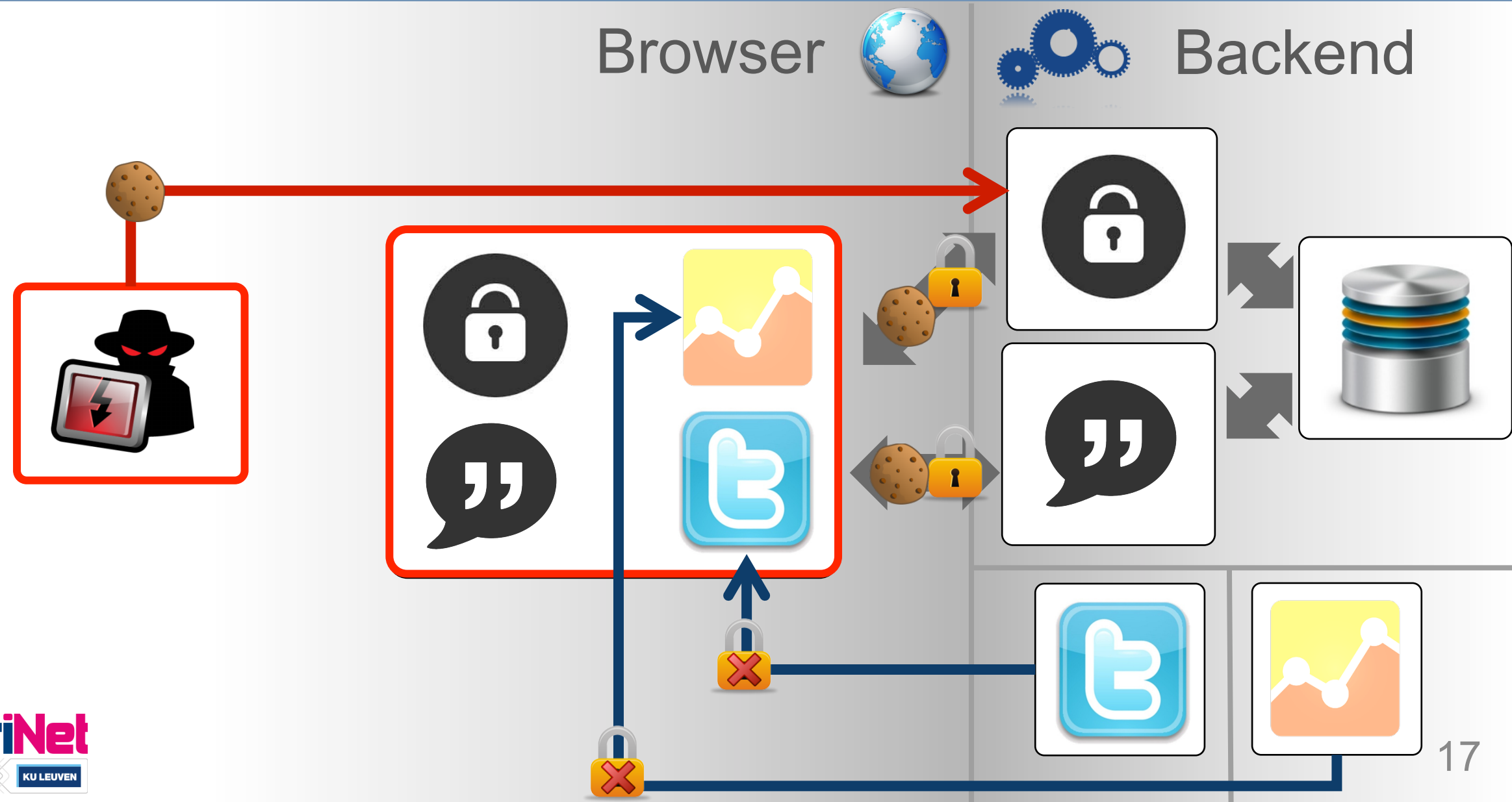
57% of HTTPS sites have no mixed content inclusions

DistriNet iMinds KU LEUVEN

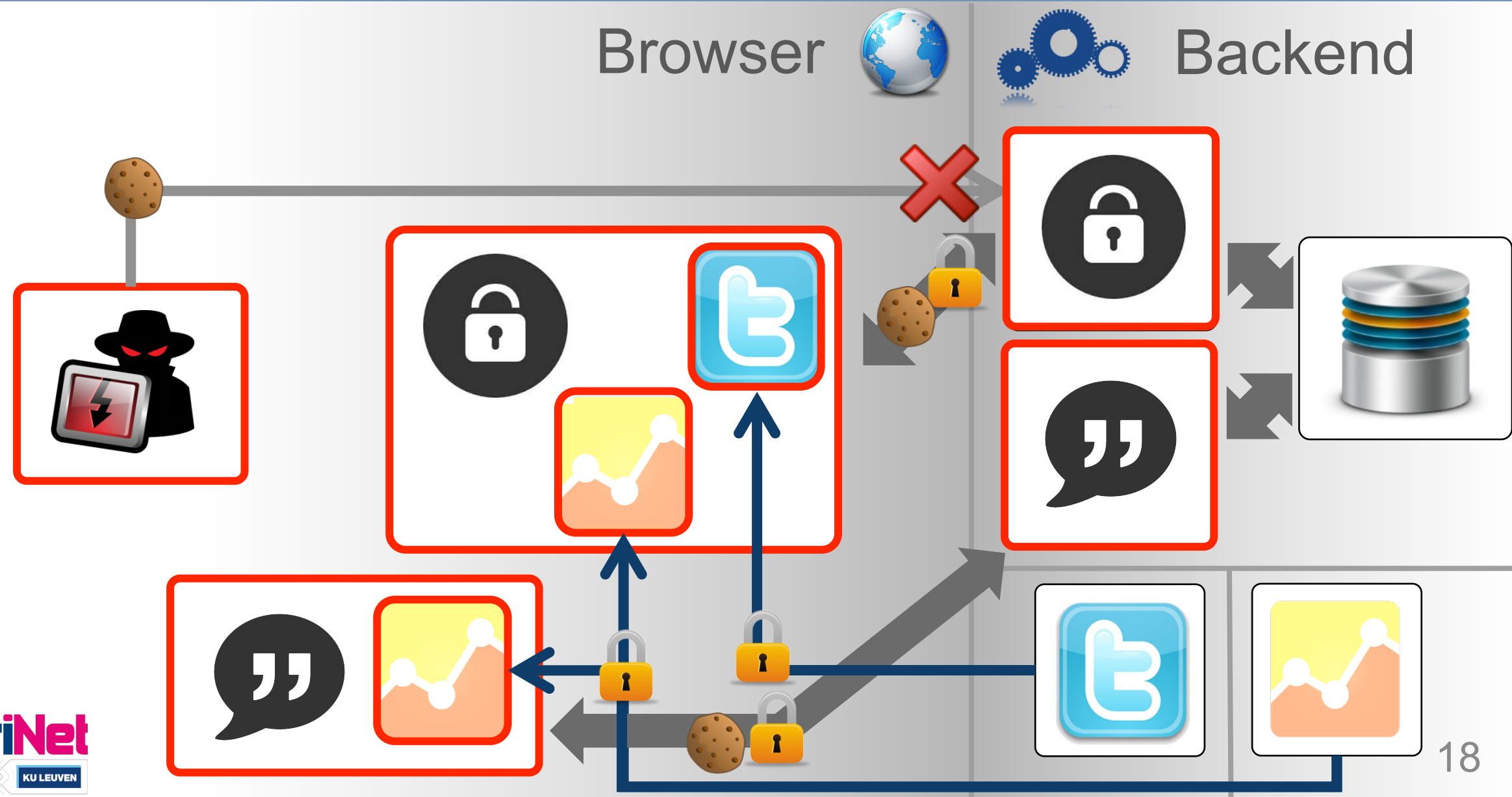
Mixed Content Inclusions



Violating Context Isolation



example.com Revisited



Challenges for this Session

- Compartmentalization using origins
  - Leverage the same-origin policy to isolate sensitive parts
- Sharing information and authentication
  - Share authentication information between contexts
  - Interact and exchange information between contexts
- Managing third-party code inclusion
  - Managing the risk associated with potentially untrusted code
  - Preventing mixed-content warnings
- Communication with the backend
  - Enable legitimate communication from HTML and JavaScript
  - Mitigate illegitimate requests from untrusted contexts

Compartmentalization

- Separation based on origin
  - Naturally enforced by the Same-Origin Policy
  - Allows you to separate sensitive parts and non-sensitive parts
  - Prevents unintended sharing of information
  - Prevents escalation of successful attack

Origin

The triple <scheme, host, port> derived from the document's URL. For *http://example.org/forum/*, the origin is *<http, example.org, 80>*

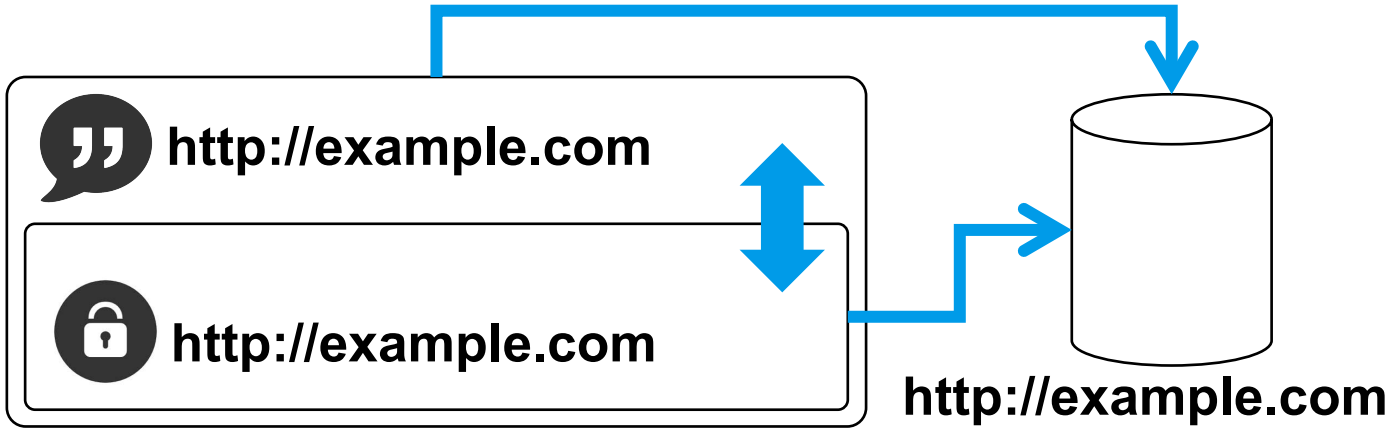
Same-Origin Policy

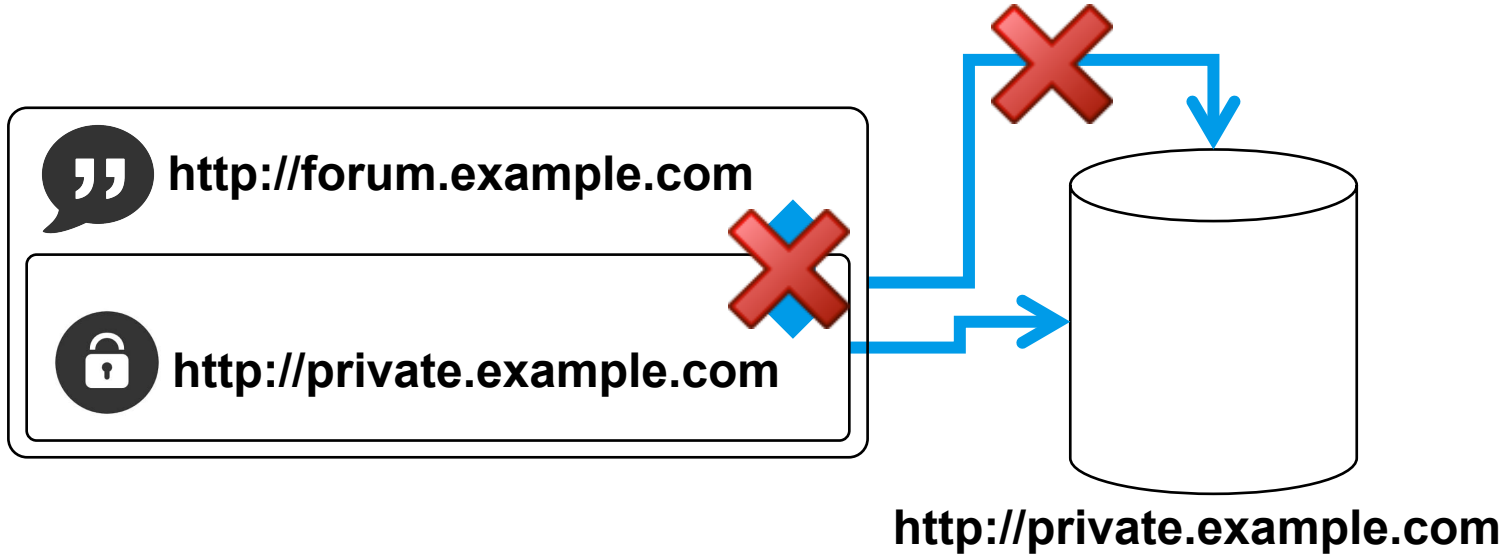
Content retrieved from one origin can freely interact with other content from that origin, but interactions with content from other origins are restricted

Examples of the Same-Origin Policy

Same-Origin Policy

Content retrieved from one origin can freely interact with other content from that origin, but interactions with content from other origins are restricted

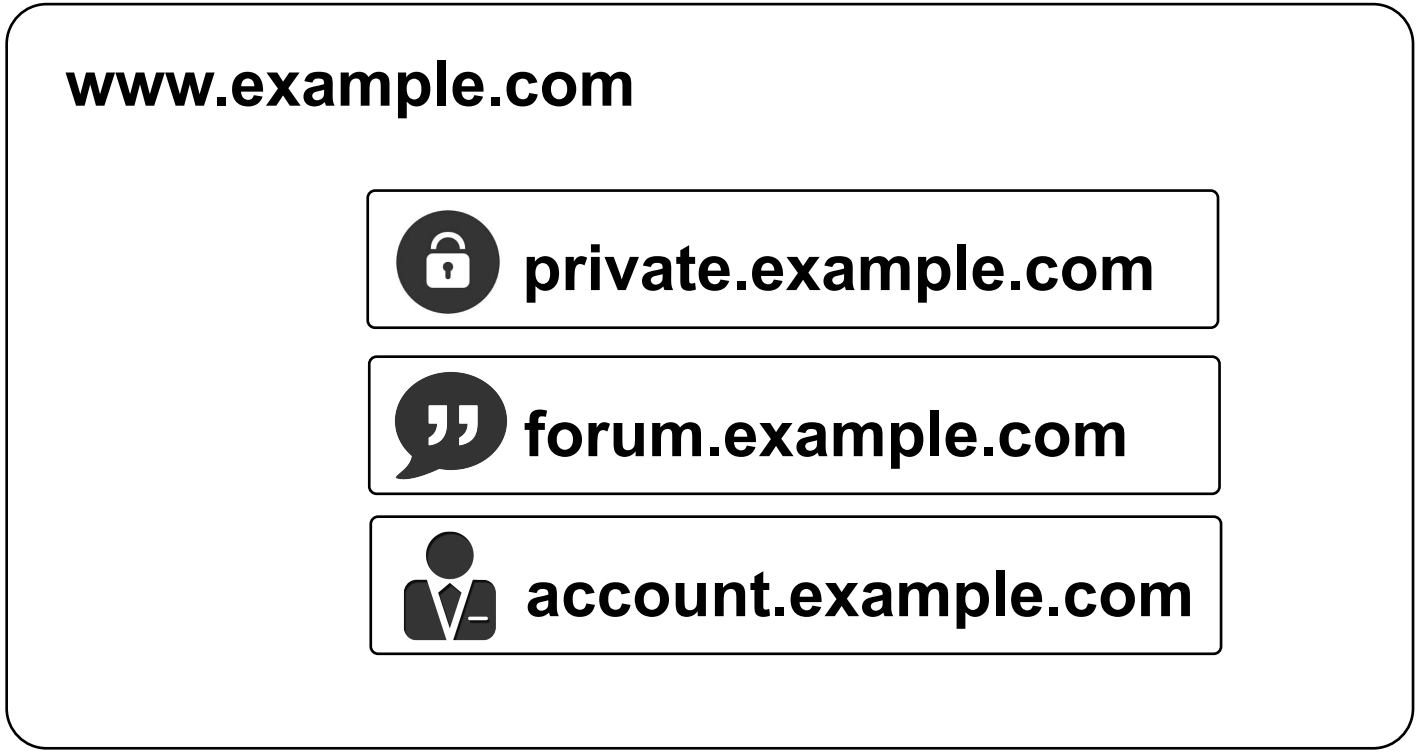




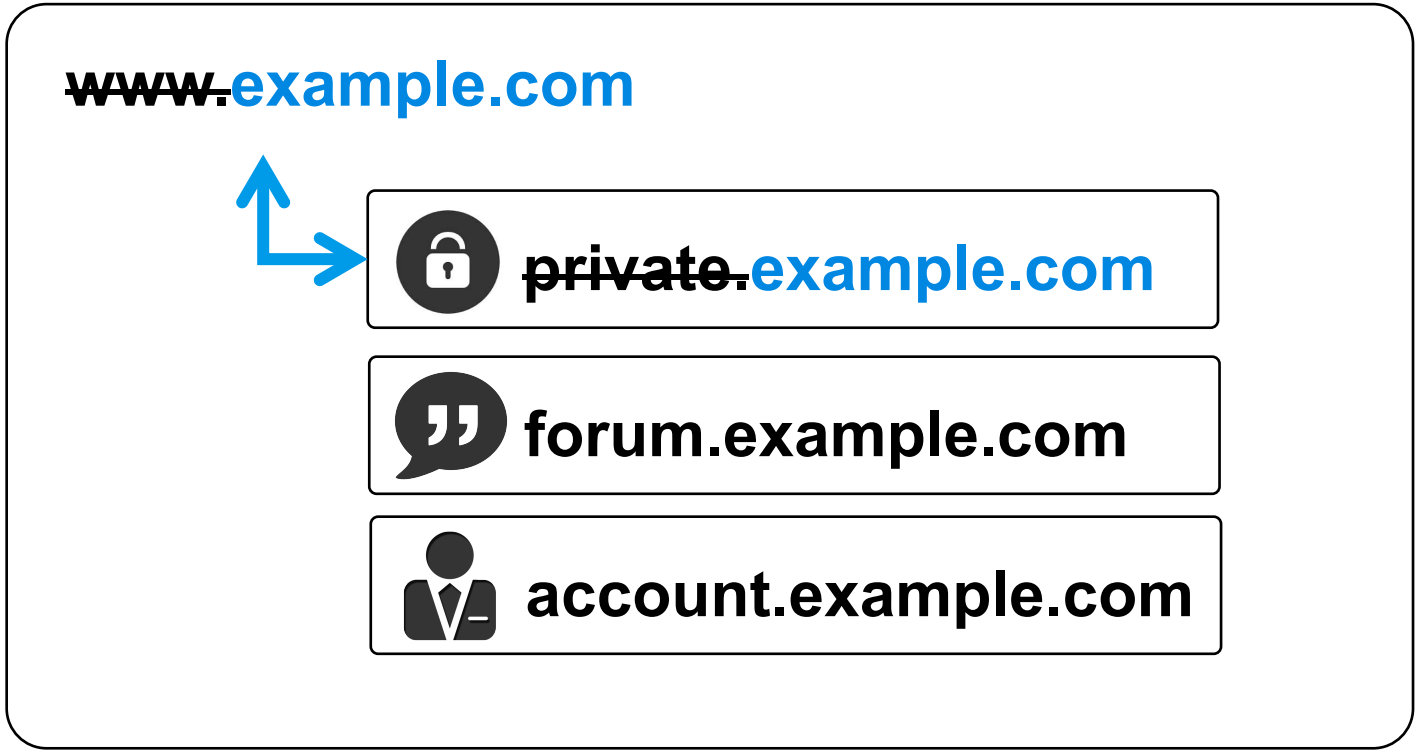
Domains vs Subdomains

- Subdomains
  - E.g. ***private.example.com*** vs ***forum.example.com***
  - Considered different origin
  - Possibility to relax the origin to ***example.com*** using *document.domain*
  - Possibility to use cookies on ***example.com***
- Completely separate domains
  - E.g. ***private.example.com*** vs ***exampleforum.com***
  - Considered different origin, without possibility of relaxation
  - No possibility of shared cookies

Subdomains and Domain Relaxation



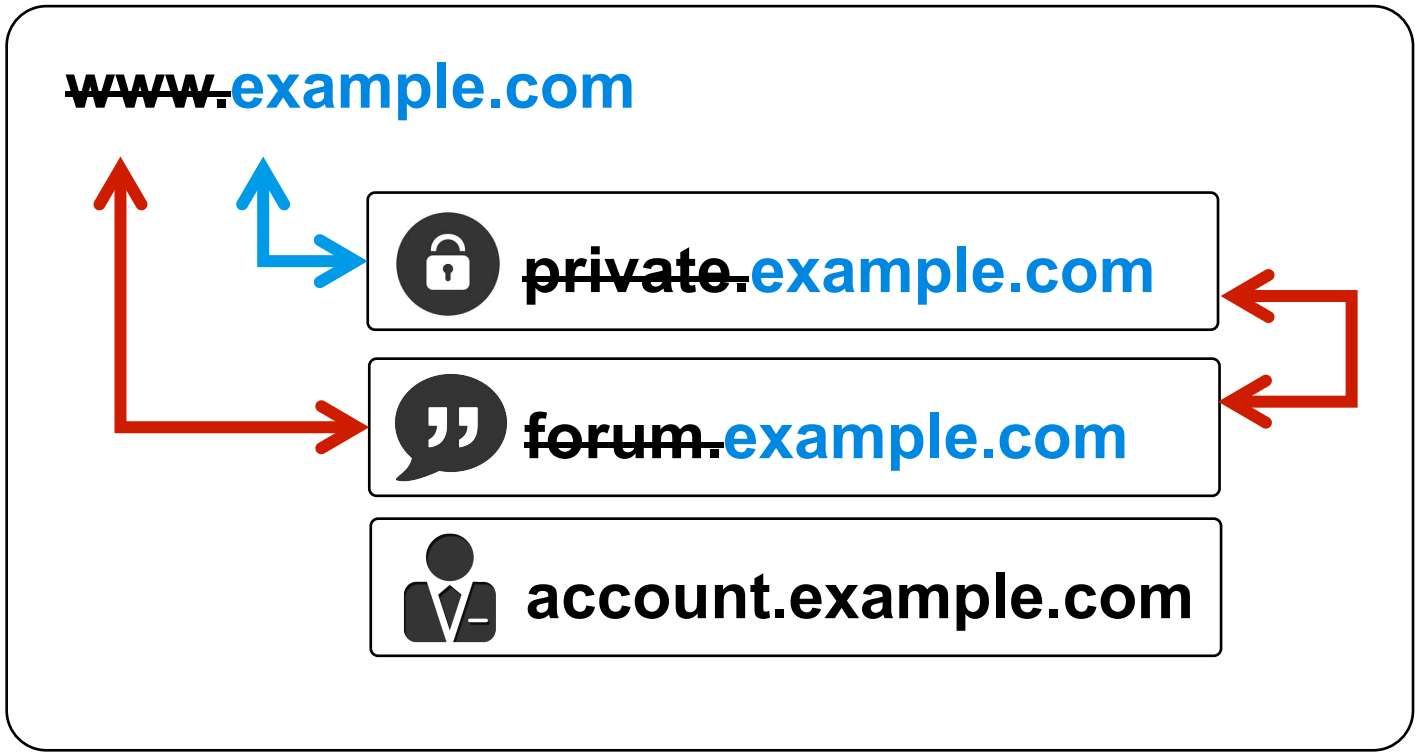
Subdomains and Domain Relaxation



Domain Relaxation

```
document.domain = "example.com";
```

Subdomains and Domain Relaxation



Domain Relaxation

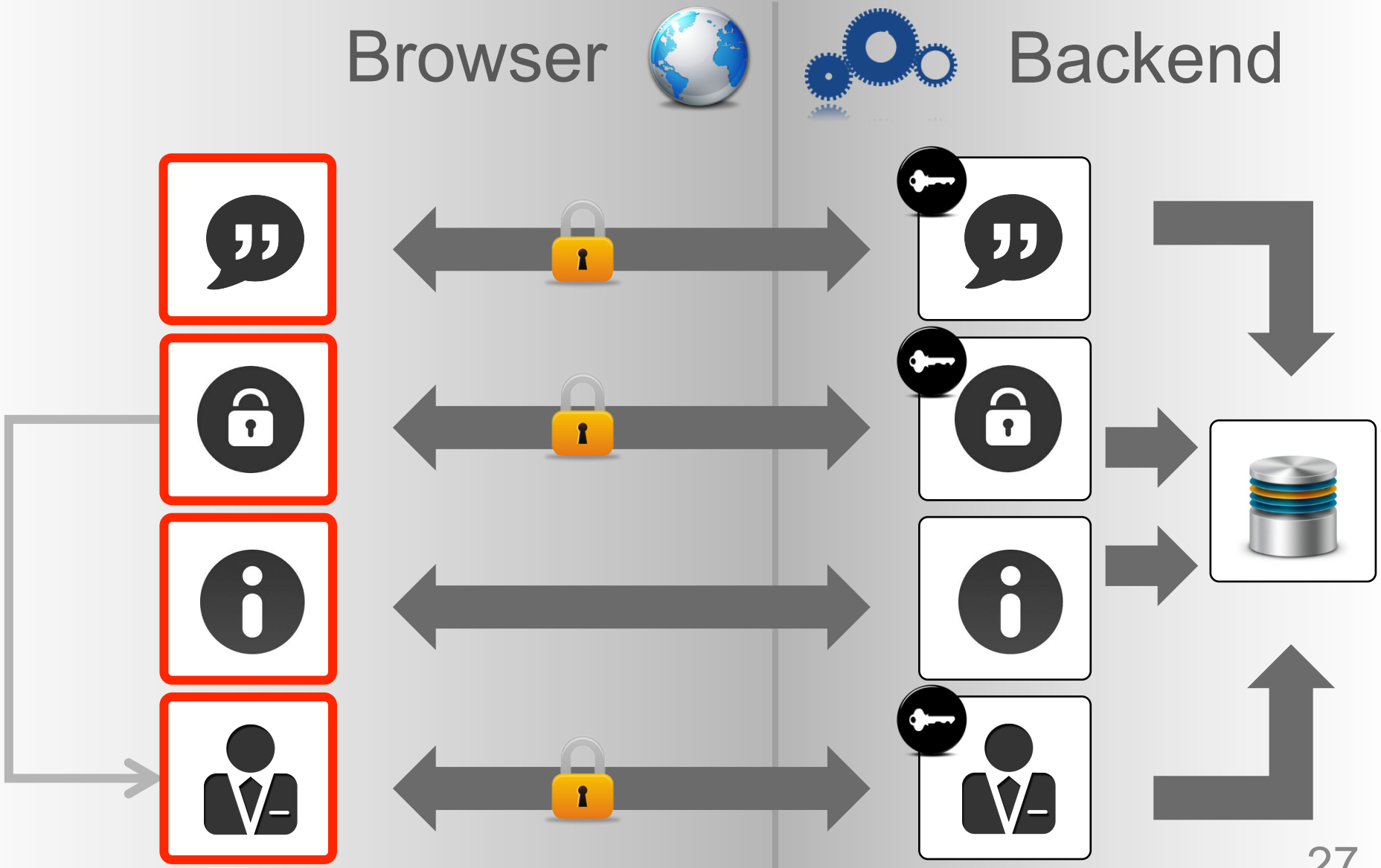
```
document.domain = "example.com";
```

Compartmentalizing *example.com*

	 Public Information	 Private Customer	 Account Management	 Public Forum
Sensitive Content	no	yes	yes	no
Requires authentication	no	yes	yes	yes
Deploy over HTTPS	preferable	yes	yes	yes
Needs cooperation	no	account	private	no
Origin	http://www.example.com	https://account.example.com		
		https://private.example.com	https://exampleforum.com	

DistriNet iMinds KU LEUVEN

Compartmentalizing *example.com*

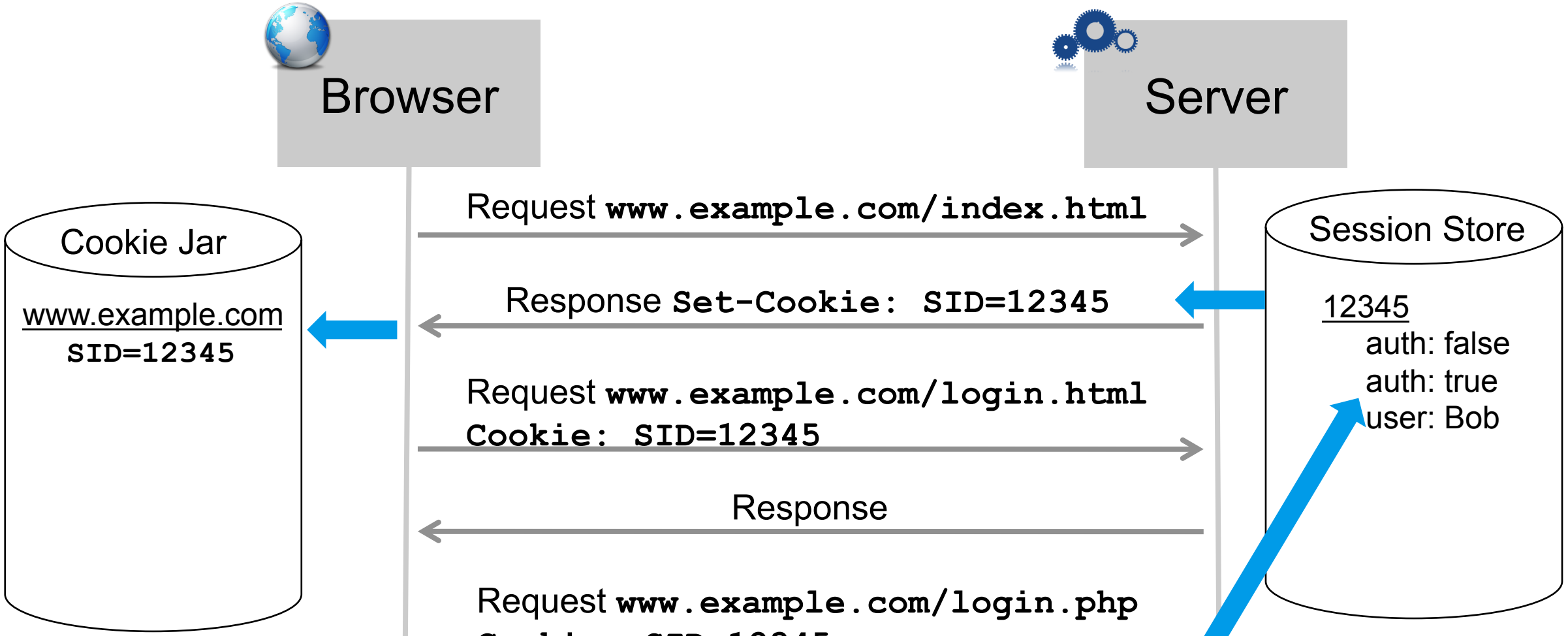


Authentication on the Web

- Typical authentication consists of two steps
  - Entity authentication
  - Maintaining the session associated with the authenticated user

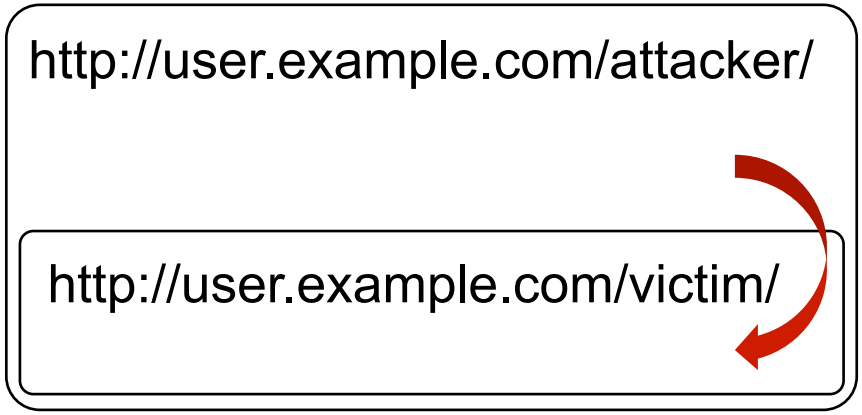
- Entity authentication
  - Exchanging username and password
  - Challenge/response systems are also used
- Session management
  - De facto standard is cookie-based session management
  - Cookie contains unique identifier, associated with server-side state

Cookie-based Session Management



Modifying Cookie Behavior

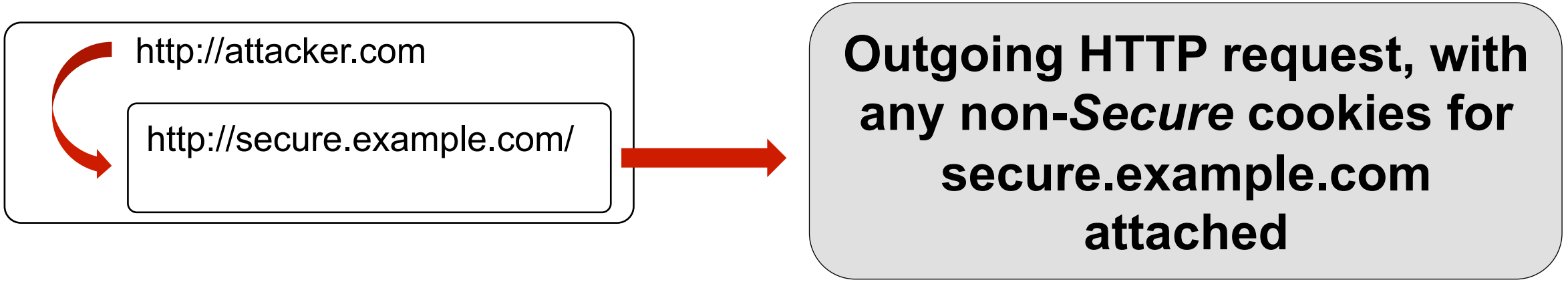
- Domain
  - Allows to broaden the applicability of the cookie
  - E.g. *example.com* applies cookie to **.example.com*
- Path
  - Associates a cookie with a specific path
  - E.g. */admin/* associates a cookie with */admin/**
  - Conflicts with the same-origin policy
- HttpOnly
  - Restricts a cookie from being accessed through JavaScript



The SOP allows direct access to the iframe, exposing *document.cookie*

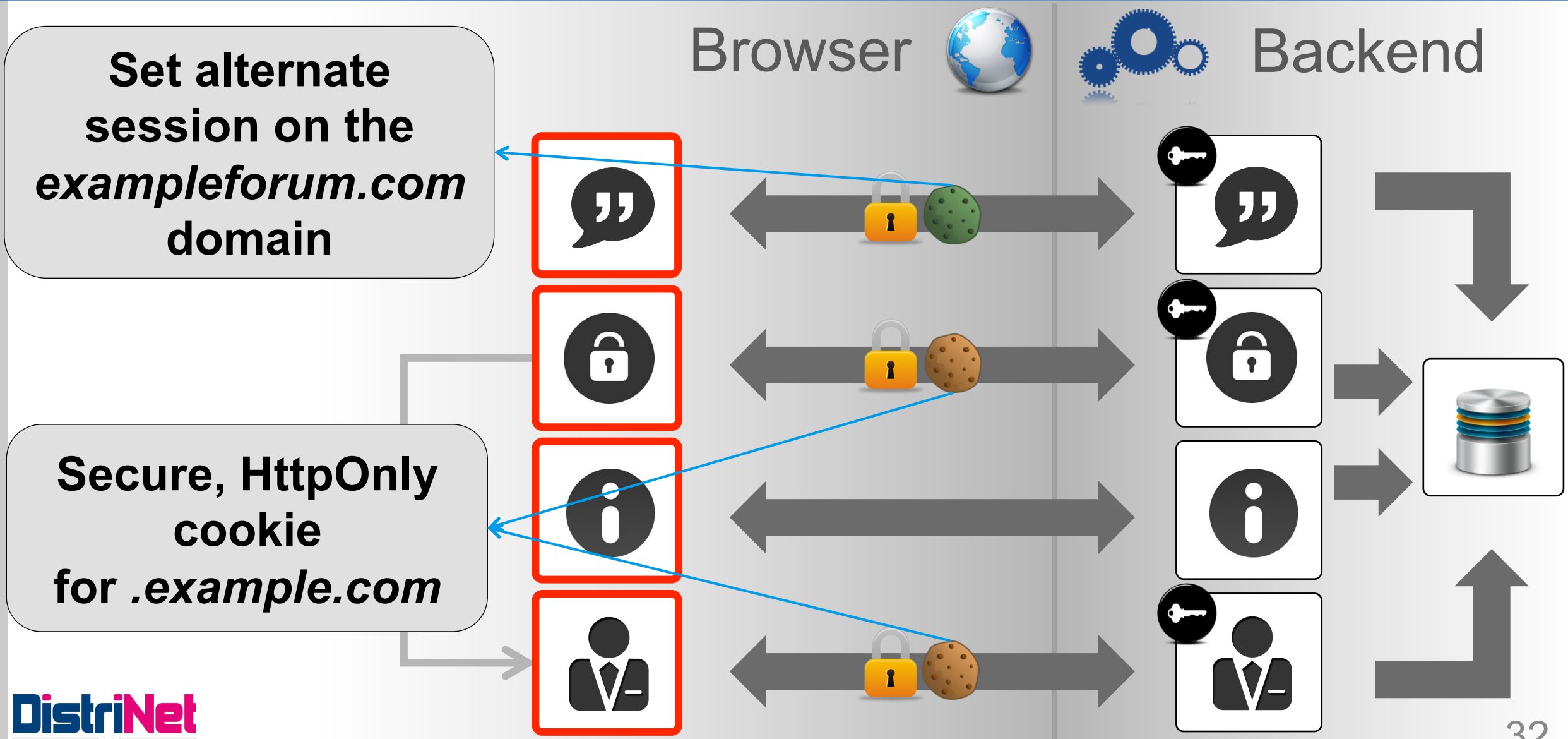
Cookies and HTTPS deployments

- Why the *Secure* flag matters
  - Cookies are associated with a domain, not an origin
  - No separation between cookies used on HTTP and HTTPS requests



- Use separate cookies for HTTP and HTTPS
  - Associate different security levels to each cookie
  - Require HTTPS cookie to be present for sensitive operations

Sharing Authentication in *example.com*



Interaction between Contexts

- Related contexts
  - Documents can open popup windows, embed frames, etc.
  - Related cross-origin contexts are isolated by default
  - Limited interactions possible (navigation, messaging APIs, …)
- Navigation
  - Navigate child frame to different resource
  - Navigate parent frame, reloading the entire document
- Exposed APIs
  - Prime example: ***Web Messaging API***, to support interaction

Web Messaging API

- Messaging mechanism between contexts
  - Used for iframes, Web Workers, etc.
  - Event listener for receiving messages (opt-in mechanism)
  - API function for sending data (text, objects, etc.)

- Security considerations
  - Specify origin of receiver to prevent leaking of content
  - Check origin of sender to prevent malicious use
  - Validate incoming content before using data to prevent injection attacks

Web Messaging API

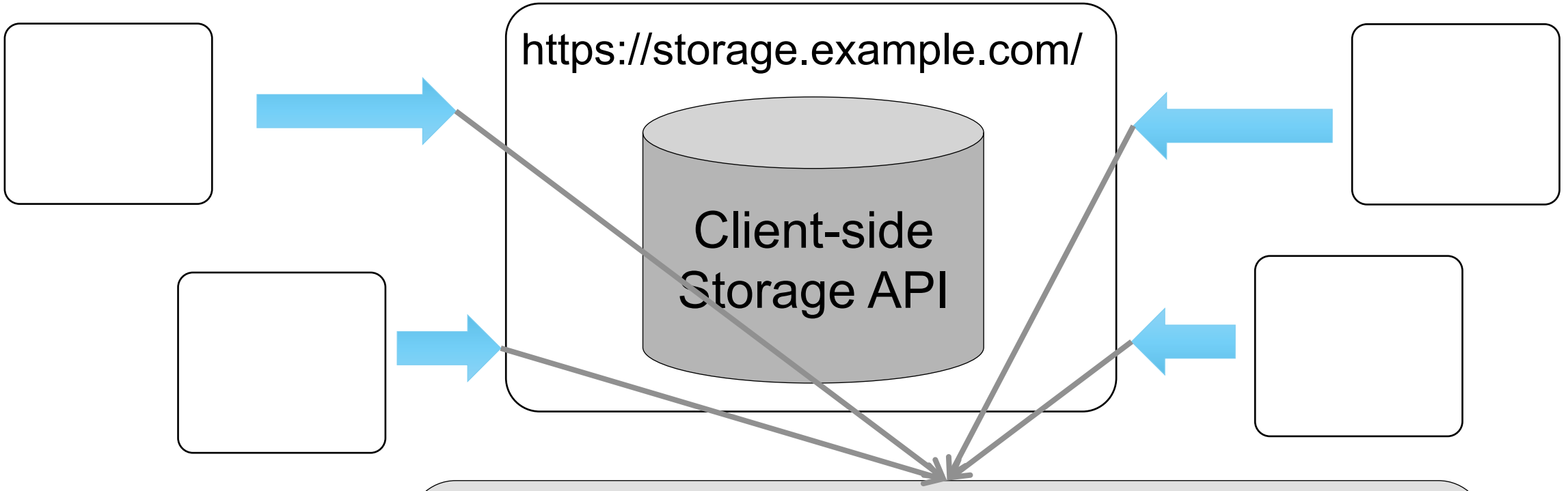
Sending Messages

```
myframe.postMessage(data,'http://test.example.com');
```

Receiving Messages

```
var handler = function(event) {
  if(event.origin ==
    'http://www.example.com') {
      alert(event.data);
  }
}
window.addEventListener('message', handler, false);
```

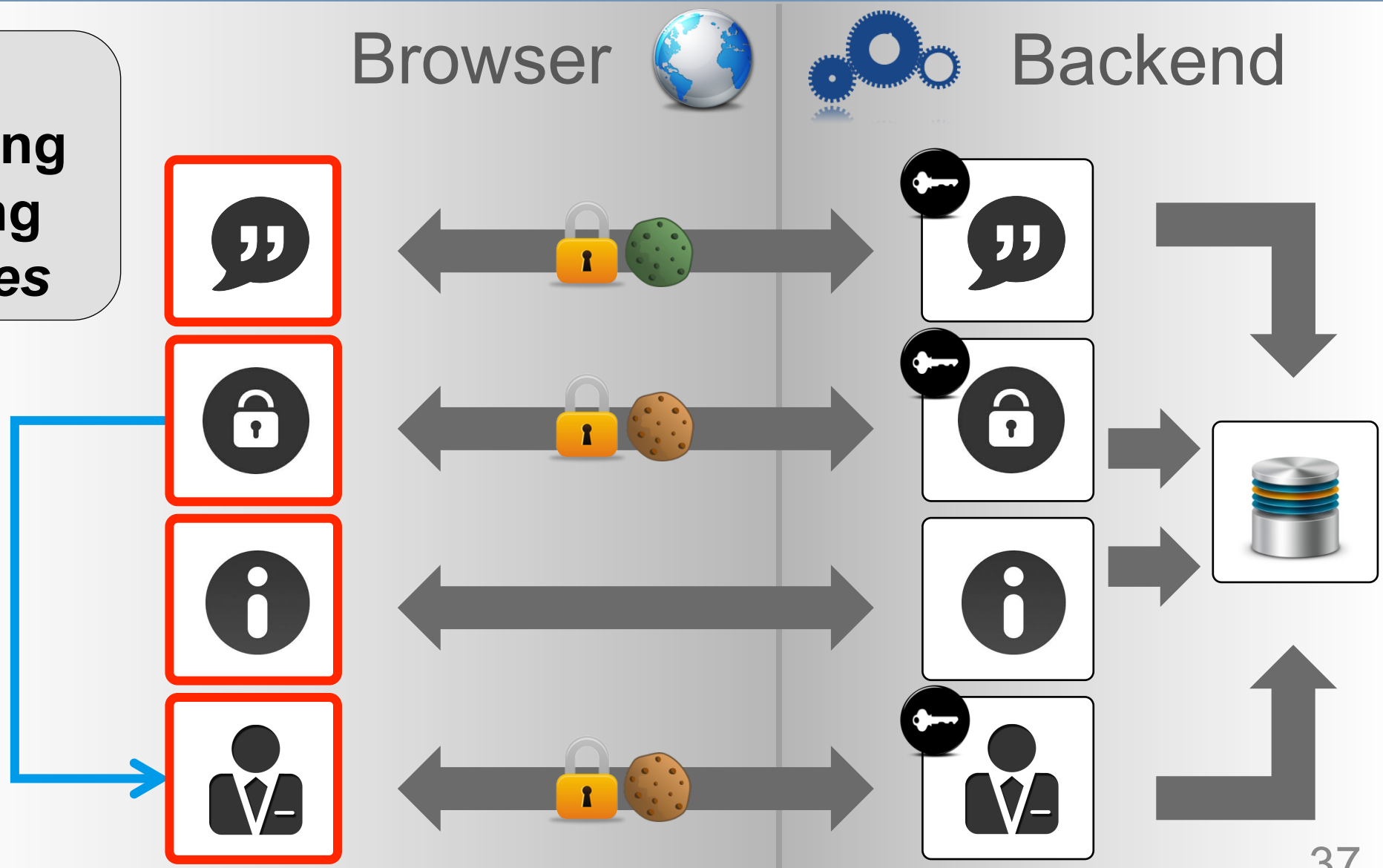
Example: a Client-side Storage Facility



Accessing local storage through Web Messaging allows enforcing access control and content inspection

Interaction in *example.com*

Exchange information using Web Messaging between *iframes*



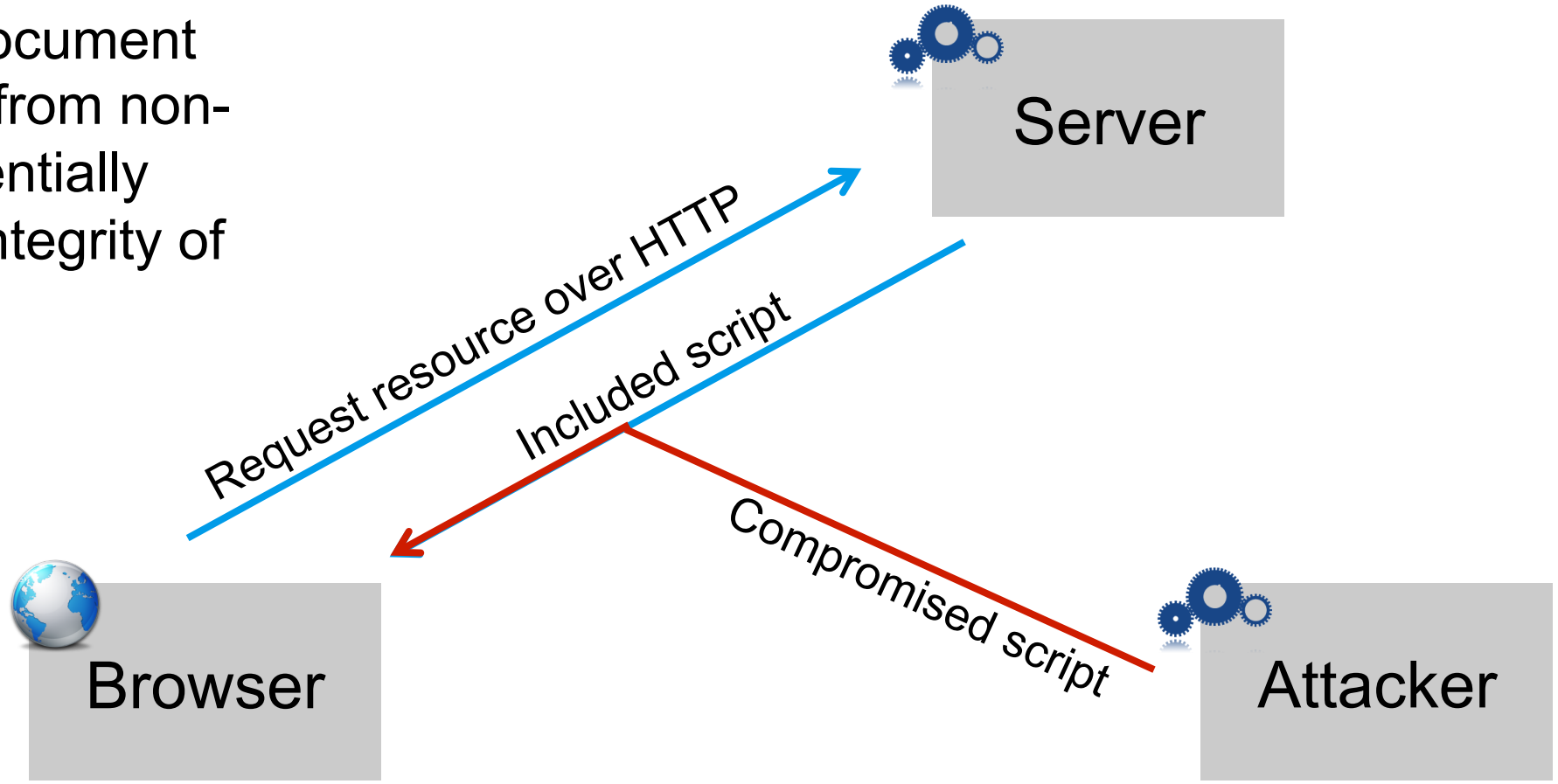
Including Remote Content

- Types of remote content
  - Images
  - JavaScript
  - CSS Styles
  - HTML documents
  - SVG images
  - Audio/video
  - Plugin content (Flash, Java)
  - …
- Including remote content
  - Identified by a URL
  - Fetched by the browser, and subsequently integrated
  - For active content (e.g. JavaScript), the included code is typically executed in the context of the including page

Mixed Content Problems

Mixed Content Inclusion

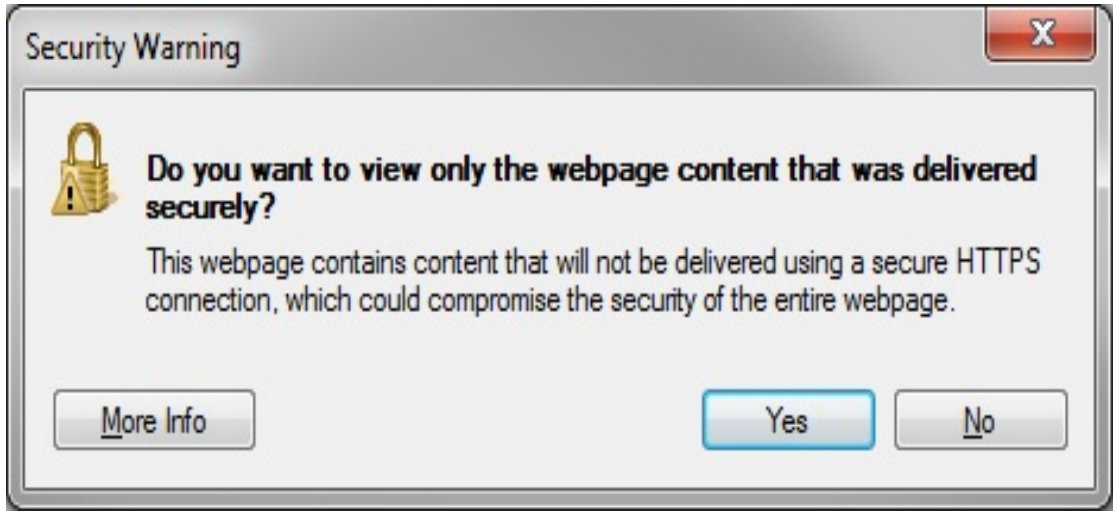
When an HTTPS-document includes resources from non-HTTP sources, potentially compromising the integrity of the document

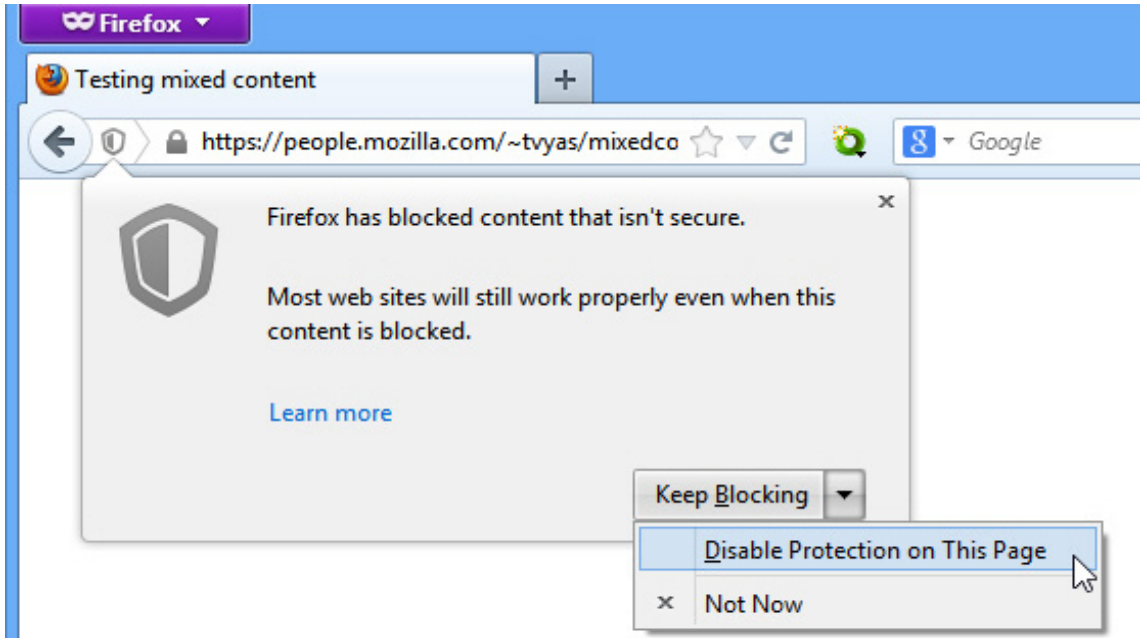


Loaded HTTPS page which requires additional resources

Solving Mixed Content Problems

- Browsers blocking mixed content inclusion
  - IE 7 started with prompting users, other browsers are following
  - **Active mixed content** is typically blocked, passive content is allowed





- Localize remote resources
  - Host remote resources locally within the application's HTTPS domain

Integration of Remote Code

- Two mechanisms to integrate code
  - Directly including JavaScript code using the <*script*> tag
  - Including code through an *iframe*, which hosts a separate document
- Scripts
  - Straightforward integration in the context, without restrictions
  - Violates the security boundaries of a document
- Iframes
  - Depending on the origin, the SOP restrictions apply
  - Preserves the security boundaries, but may hinder interaction

Script-based Content Integration

- No security boundaries offered by browser
  - Combination with remote providers is potentially dangerous
  - Full access to the client-side context, including local resources

- Existing techniques to constrain scripts
  - Localizing scripts → requires effort to update
  - Safe subsets of JavaScript → requires compatibility with existing scripts
  - Browser-based sandboxing → requires modifications to the browser
  - Server-side rewriting → requires control over the scripts
  - JavaScript-based sandboxing → upcoming technology

Iframe-based Content Integration

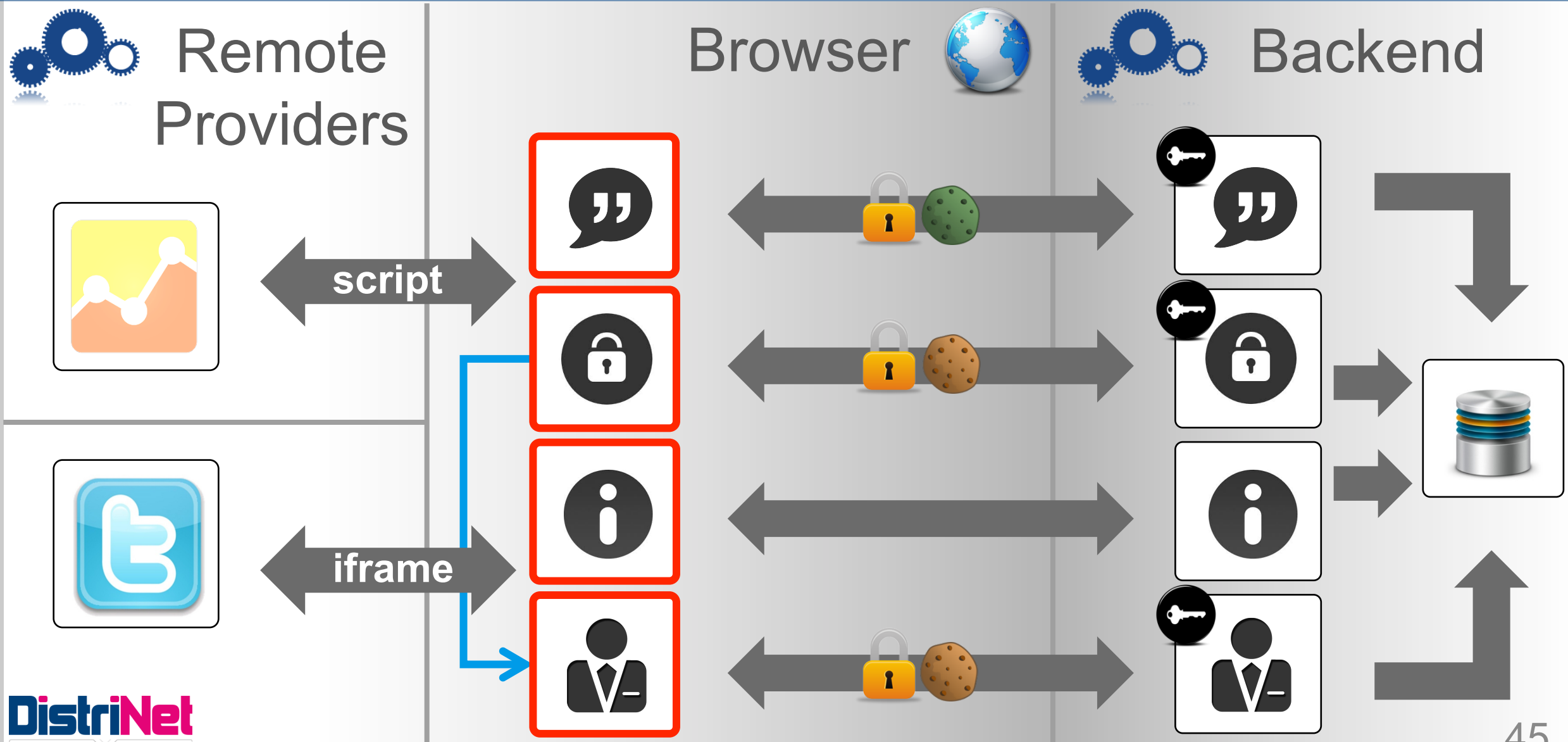
- Iframes are controlled by the same-origin policy
  - Documents with different origins are isolated by the SOP
  - Well-suited to integrate separate components (e.g. advertisements)
  - More difficult to achieve dynamic interaction

- HTML5 introduces the *sandbox* attribute
  - Gives coarse-grained control over capabilities in an iframe
  - Supports disabling scripts, plugins, forms, etc.
  - Supports a unique origin, alienating the iframe from any other origin
  - Well-suited for the integration of untrusted content

Best Practices for Integrating Code

- If possible, isolate the content in an iframe
  - Use the *sandbox* attribute to enforce even more restrictions
  - Especially true for untrusted content (e.g. user-provided)
- Only include code from trusted providers
  - Google often provides mirrors of popular libraries
- Localize scripts for crucial applications
  - Keep scripts regularly up-to-date
  - Perform code reviews of the differences between versions

Remote Code in *example.com*



Interacting with Remote Services

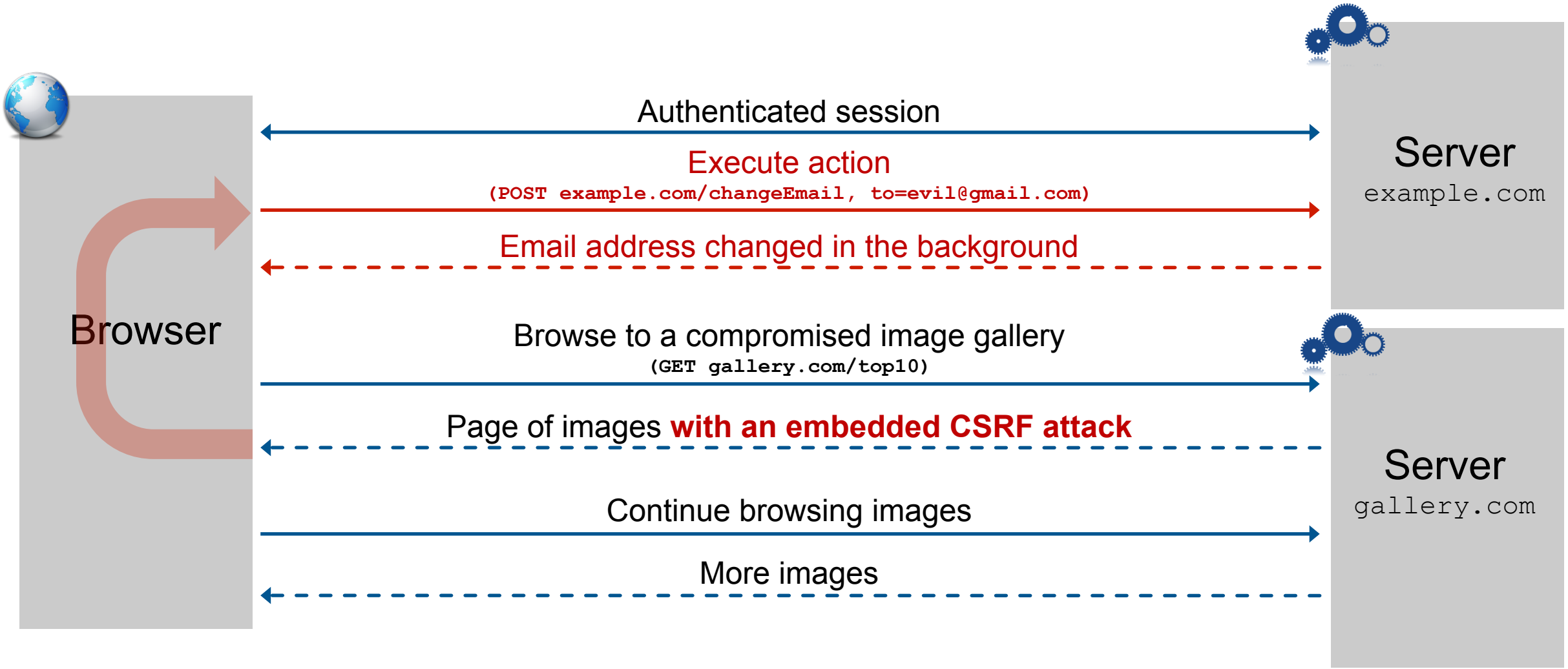
- Ways to interact remotely
  - Triggered from HTML elements (image loads, form submissions, …)
  - Programmatically from JavaScript (XMLHttpRequest)
  - Using alternative protocols (Web Sockets, WebRTC, …)

- Challenges with remote interaction
  - Difficult to determine which context a request originated from
  - Difficult to determine if a request was intended by the user

HTML-based Remote Interaction

- Several types of requests can be triggered
  - GET requests from <*img*>, <*script*>, …
  - POST requests with control over body content from <*form*>
- Not affected by the Same-Origin Policy
  - GET and POST requests can be sent to other origin
  - Browser attaches available cookies to the request
- Session cookies are implicit authentication!
  - Results in an attack known as Cross-Site Request Forgery

Cross-Site Request Forgery (CSRF)



Mitigating Cross-Site Request Forgery

- Mitigation techniques need to be explicitly present
  - Token-based approaches
  - Origin header

TOKEN-BASED APPROACH

example.com

```
<form action="submit.php">
  <input type="hidden" name="token"
    value="qasfj8j12adsjadu2223" />
  …
</form>
```

Programmatic Remote Interaction

- Sending requests with XMLHttpRequest
  - Supports different types of requests
  - Possibility to modify/manipulate “safe” headers
  - Response can be processed from within JavaScript

SENDING REQUESTS

```
var url = “http://test.example.com/api.php”;
var req = new XMLHttpRequest();
req.open("GET", url, true);
req.onload = function(e) { … }
req.send();
```

XMLHttpRequest and the SOP

- Same-origin requests
  - No restrictions imposed on the use of XMLHttpRequest
  - Custom headers, use of credentials, etc.
- Cross-origin requests
  - Required to enable remote interaction (e.g. APIs) without hacks
  - Enables capabilities not found in traditional HTML (e.g. PUT, DELETE)
  - Legacy server code does not expect such cross-origin requests
- New security policy: Cross-Origin Resource Sharing

Cross-Origin Resource Sharing (CORS)

- Enables client-side cross-origin requests
  - Opt-in mechanism to grant other origins access to certain resources
  - Allows the easy use of online APIs without hacks
- Preventing additional attack vectors
  - Configurable security policy to determine who can access response
  - Preflight request to approve “dangerous” requests up front
  - Attacker capabilities with CORS largely correspond to HTML elements
- Already used beyond XMLHttpRequest
  - Regulating access to cross-origin HTML elements (canvas, …)

Cross-Origin Resource Sharing (CORS)

Sending CORS Requests

```
var url = "http://api.provider.com/api.php";
var req = new XMLHttpRequest();
req.open("GET", url, true);
xhr.withCredentials = true;
req.onload = function(e) { … }
req.send();
```

CORS Response Headers

Access-Control-Allow-Origin: ***http://www.example.com***
Access-Control-Allow-Credentials: ***true***
Access-Control-Expose-Headers: ***APIVersion***

Sharing an API with CORS

Public CORS API (/api/public/)

- Allow wildcard origin

```
Access-Control-Allow-Origin: *
```

Restricted CORS API (/api/accounts/)

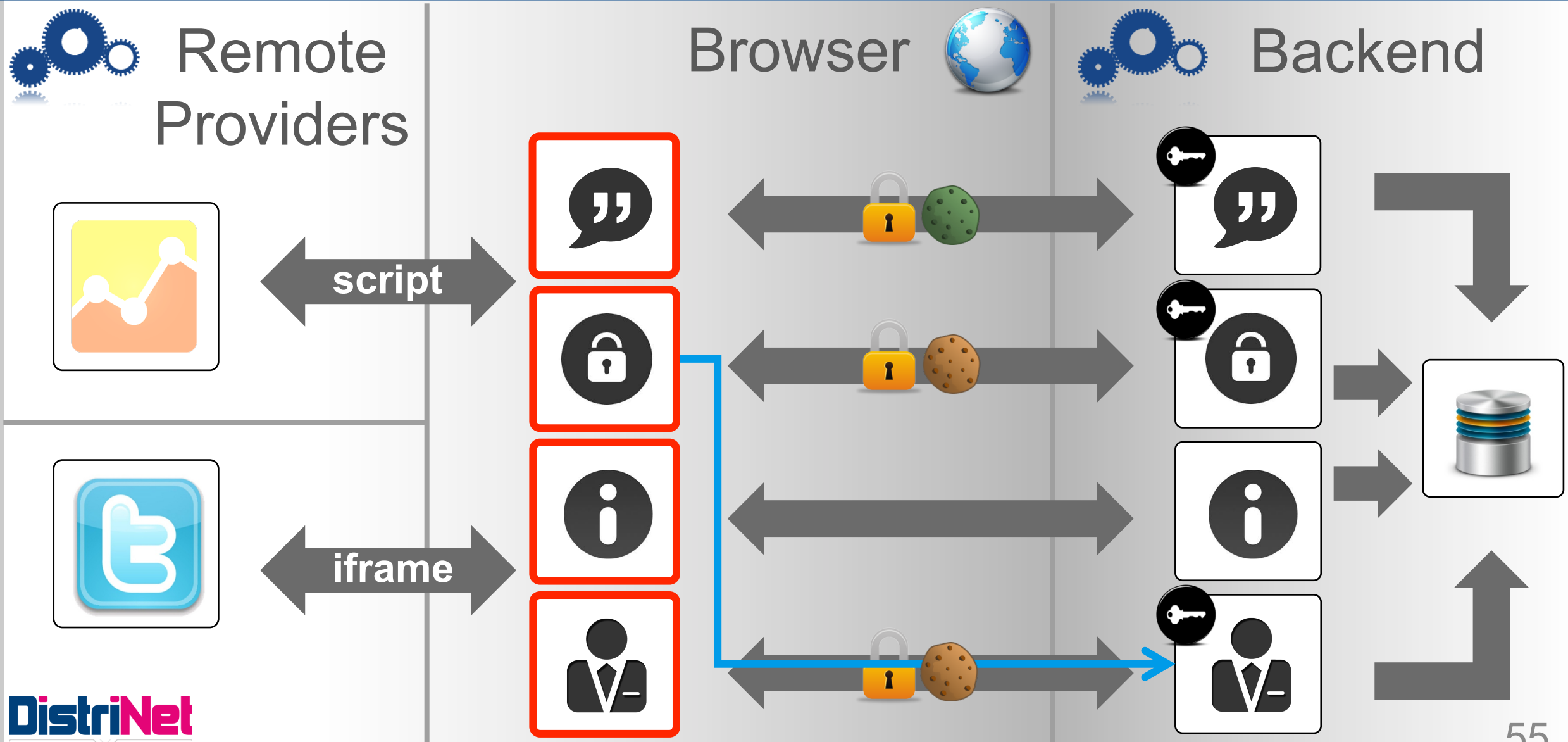
- Allow the customer area origin
- Allow the use of credentials
- Expose the X-*Version* header

```
Access-Control-Allow-Origin: https://private.example.com
Access-Control-Allow-Credentials: true
Access-Control-Expose-Headers: X-Version
```

CORS Processing Checklist

- Check origin of request
- Check used method
- Perform traditional access control
- Execute request
- Add appropriate response headers

Remote Interaction in *example.com*

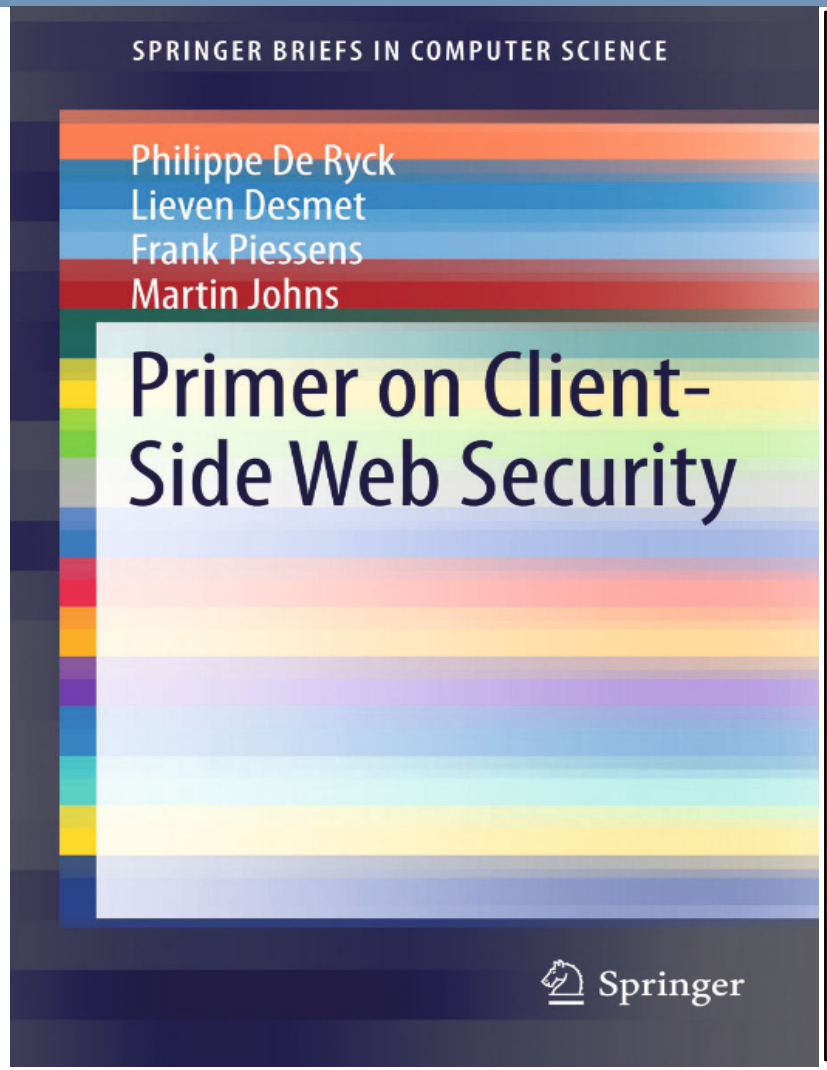


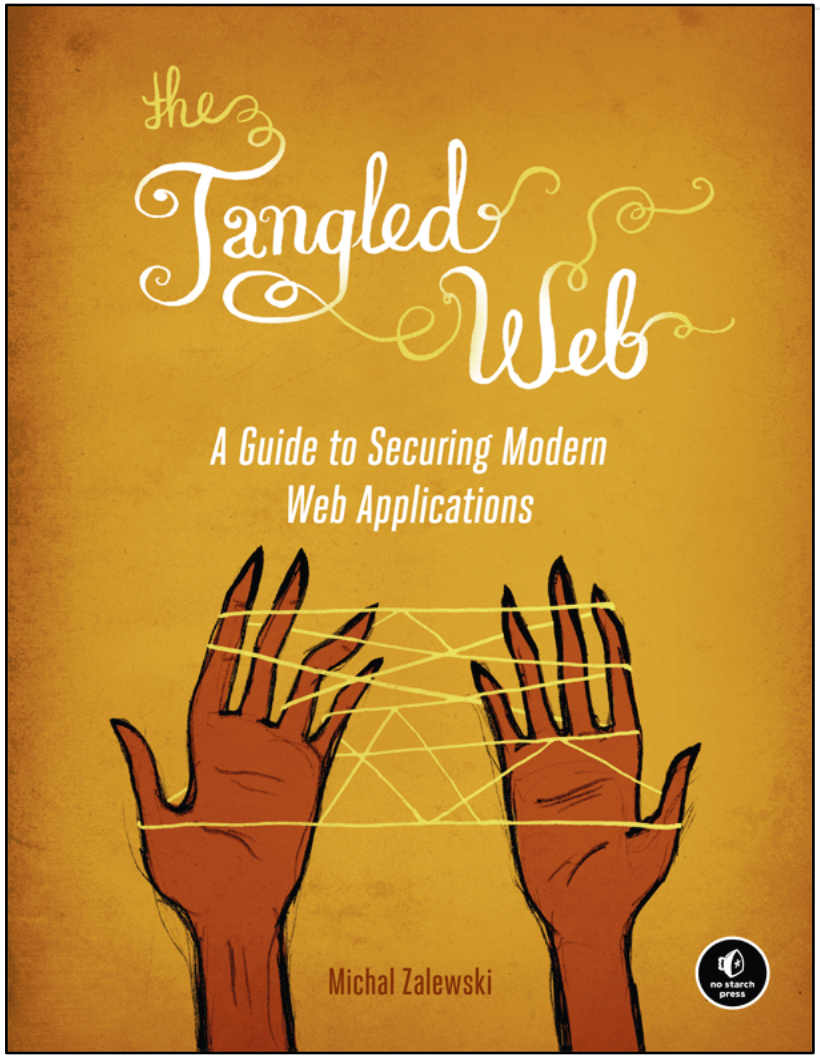
Wrap Up

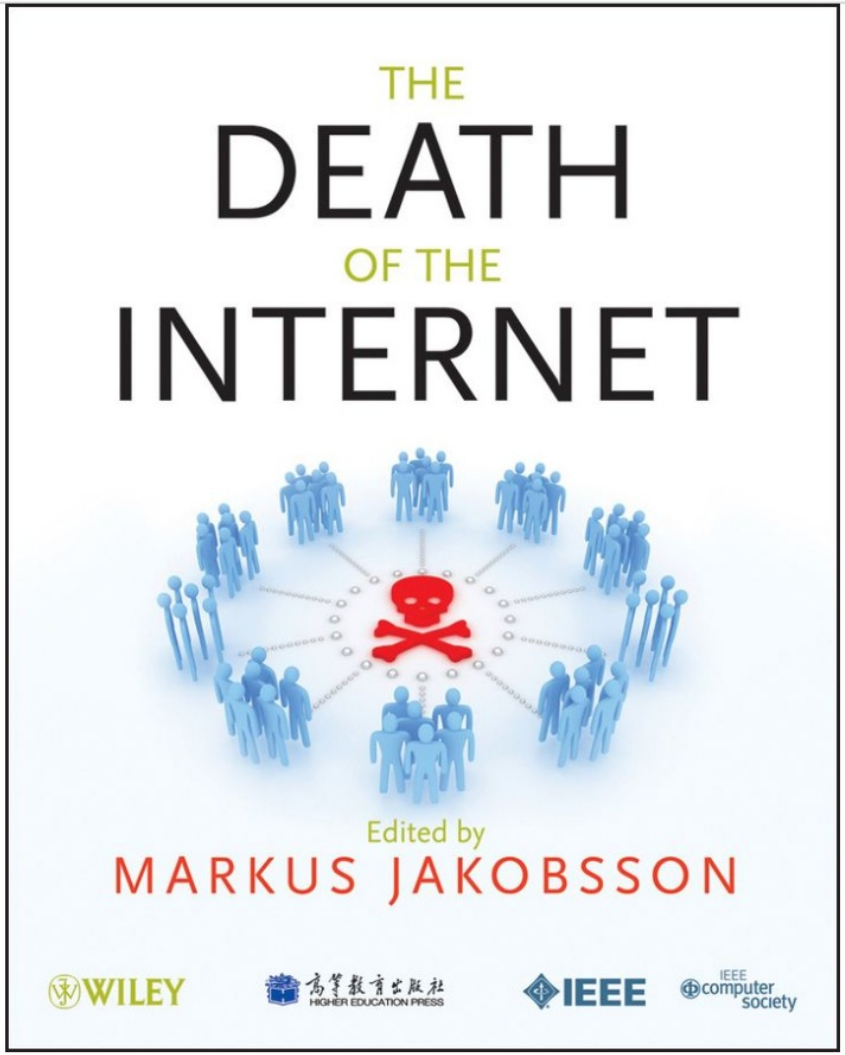
Take-home Message

- The *origin* is a core concept in web security
- Compartmentalize where possible
- Treat incoming messages as potentially untrustworthy
- Consider the trust relationship with external parties

Further Reading







The Web's Security Model

Philippe De Ryck – iMinds-DistriNet, KU Leuven

philippe.deryck@cs.kuleuven.be

DistriNet

iMinds KU LEUVEN

SecureApplication Development

secappdev.org