Bridging the Security Gap with Decentralized Information Flow Control

Adam Belay, Andrea Bittau, Dan Boneh, Pablo Buiras*, Daniel Giffin, **Amit Levy**, Ali Mashtizadeh, David Mazieres, John Mitchell, Alejandro Russo*, Amy Shen, Deian Stefan, David Terei, Edward Yang, Nickolai Zeldovich†

Stanford, †MIT, *Chalmers

Project Goal

Make it possible for programmers who are not security experts to build secure web applications

Hails, LIO/DCLabels, Safe Haskell

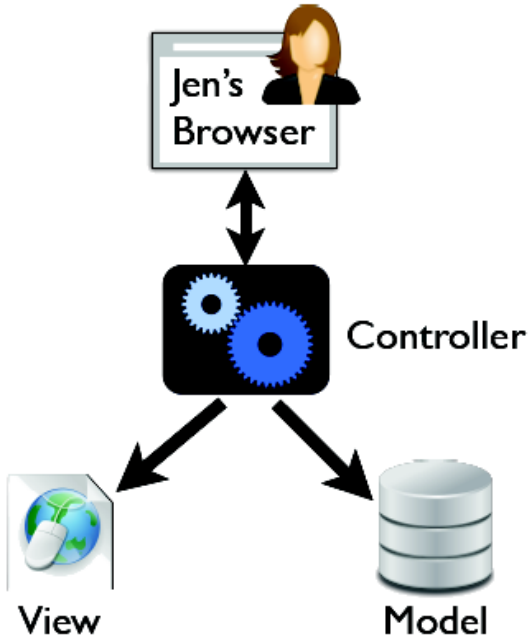
Pablo Buiras, Amit Levy, Deian Stefan, David Terei,
David Mazieres, Alejandro Russo

Outline

1. Motivation
2. High level overview of Hails
3. Mechanisms:
    1. Haskell and Safe Haskell
    2. LIO and DCLabels

The Server Side Today: Web Apps

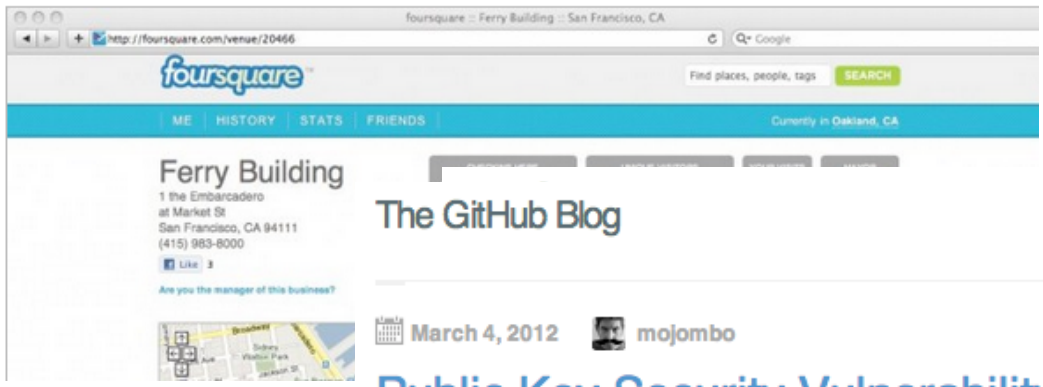
- Most apps structured around MVC (Model-View-Controller)
  - Rails, Django, Struts, .NET, others…
- Useful for compartmentalizing development



Why is the Web so &$@*ing Broken?!

Foursquare vulnerability exploited: 'private' location data captured

Jun. 30, 2010 (10:15 am) **By:** *Andy Carvell*



The GitHub Blog

March 4, 2012 mojombo

Public Key Security Vulnerability and Mitigation

Confluence Security Advisory 2012-05-17

Added by Andrew Lui [Atlassian Technical Writer], last edited by Sarah Maddox (Administrative Account) on Aug 08, 2012

This advisory discloses a **critical** security vulnerability that exists in all versions of Confluence up to and includin

- **Customers who have downloaded and installed Confluence** should upgrade their existing Confluence i
- **Enterprise Hosted customers** need to request an upgrade by raising a support request at http://support.a project.
- **JIRA Studio and Atlassian OnDemand customers** are not affected by any of the issues described in this

The Server Side Today: Web Apps

Well...

- No notion of security policies
- Ad-hoc security checks throughout applications
  - Easy to forget a check (e.g. GitHub mass assignment volnerability)
  - Extracting the policy requires looking at the **whole application**
  - Often breaking MVC abstraction

Hails: A web platform framework

Goals

- Suitable for web *platforms*
- Usable by web developers
  - Easy to write policies
  - Easy to write the rest of the app
- Deployable today
  - Change as little of the stack as possible

What are web *platforms*?

Web platforms are collections of independant apps that share date

What are web *platforms*?

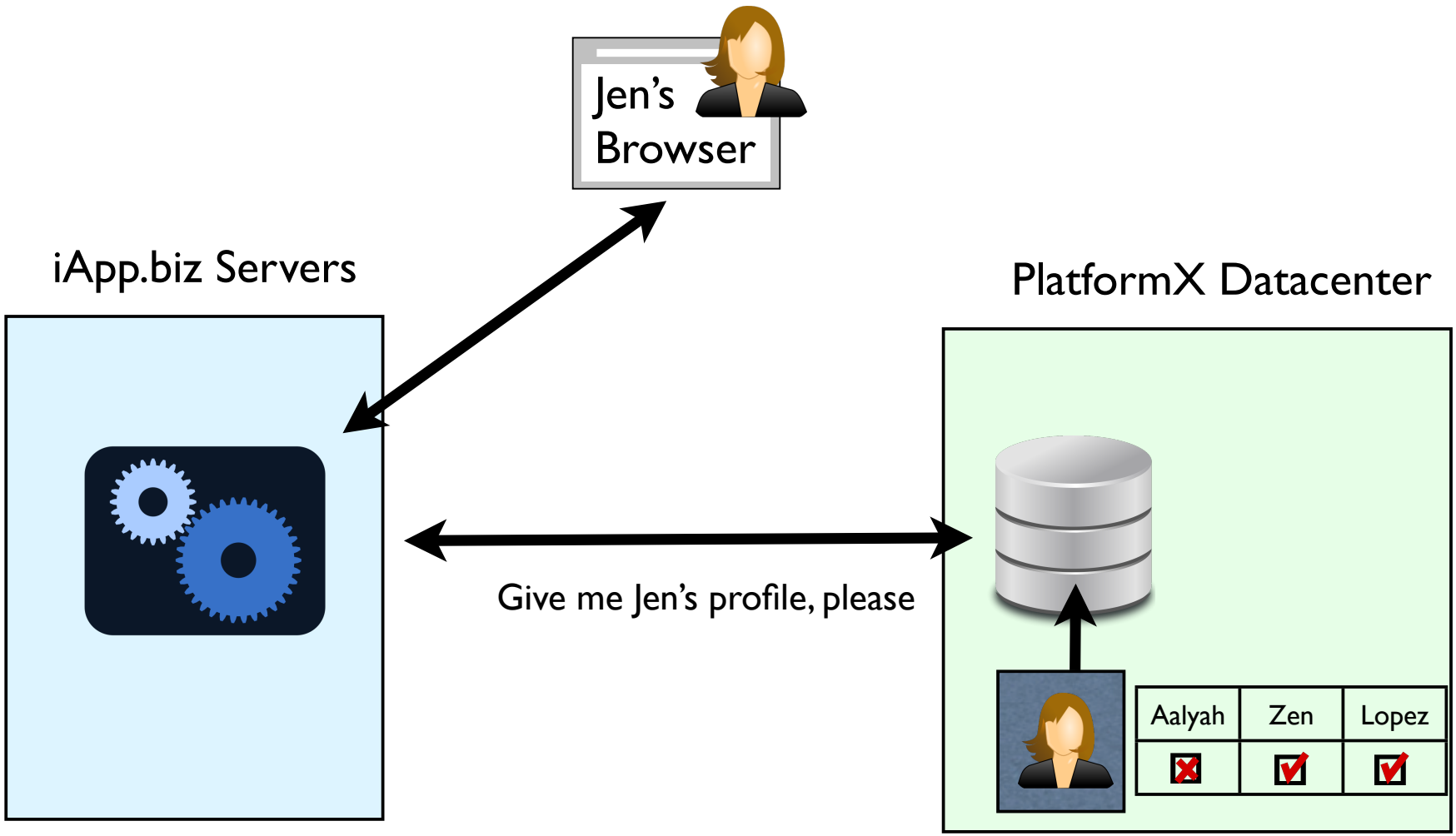
Web Apps

- Run by a single enitity
- Are developed by a single organization
- Grant all components complete access to all data

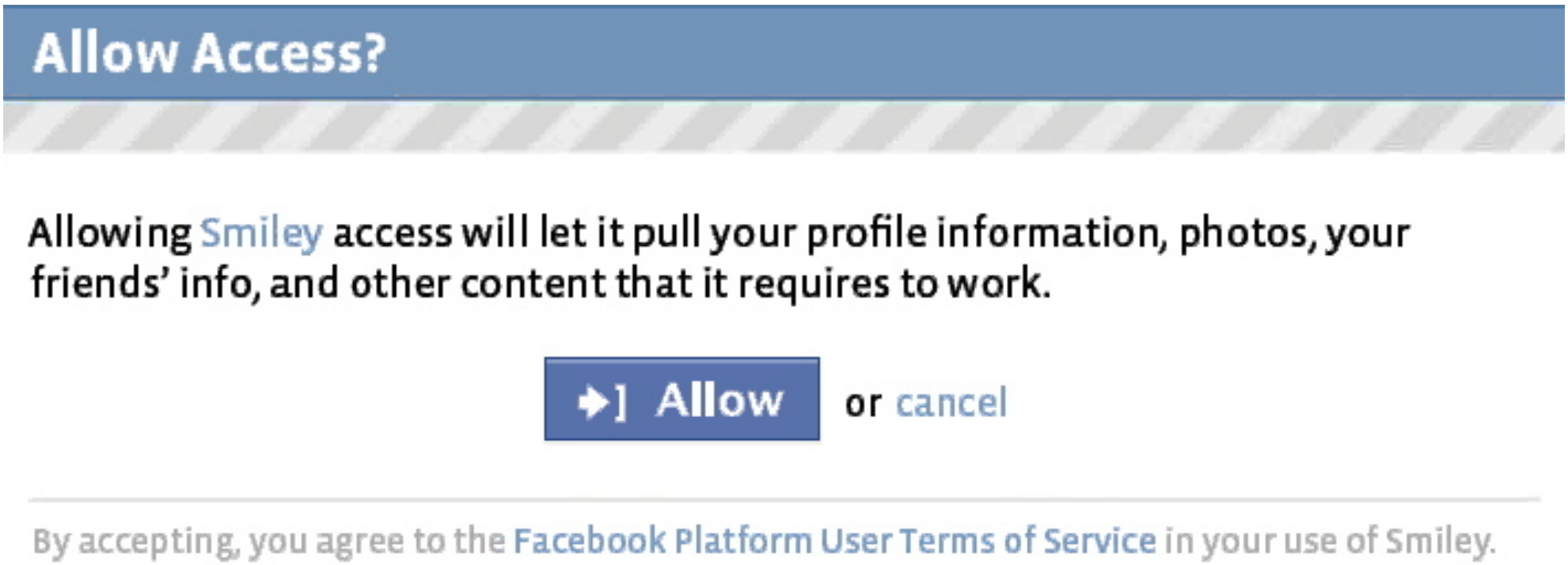
Web Platforms

- Consist of apps run by various entities
- Developed by myriad organizations not-necessarily in collaboration
- Different components have different access level to data.

The Server Side Today: Web Platforms



Current Solution



Change the hosting model

Instead of

- Developers hosting apps on in their own datacenters
- Platforms enforcing security contractually (e.g. terms of service)

Hails: A new approach

- *Platforms* host apps on their own hardware, on top of Hails
- Use information flow control to **ensures** apps obey security policies

Adding Policy to MVC

- New paradigm: Model-**Policy**-View-Controller
  - Policy specified independantly
  - No policy in the Model, View or Controller
- Hails has two types of third-party code
  - Model-Policies (MPs)
  - Provide data model and policy
  - View-Controllers (VCs)
  - Web server executables that link to MPs

Trust Model in Hails

- View-Controllers are completely untrusted
  - Includes most of the interesting functionality, like UI
- Model-Policies must only be trusted with the data they define
  - Users have to trust that they set good policies.
- Hails uses information flow control (IFC) do enforce policies on data models, end-to-end

Mechanisms

Haskell & Safe Haskell

Haskell

- Safe(ish), strongly typed, pure
  - Strict separation of side-effectful code through Monads:

```
putStrLn :: String -> IO ()
```

```
map . toLower -> String -> String
```

- Built-in code compartmentalization
  - Packages
  - Modules
- Allowed us to implement IFC as a library

Safe Haskell

An extension to GHC developed by David Terei. Included in GHC since version 7.

- Haskell has some builtin *holes* in the type system:
  - `unsafePerformIO, OverlappingInstances`
- Haskell has some holes in the module system
- Safe Haskell closes those holes:
  - -XSafe modules cannot use unsafe operations or depend on unsafe modules
  - Trustworthy modules must reside in packages that are explicitly marked trusted by admin

Mechanisms

DCLabels and LIO - Decentralized Information Flow Control (DIFC)

Information Flow Control Labels

Labels are points on a lattice with well defined $\sqsubseteq$, $\sqcap$, and $\sqcup$:

```
class (Eq l, Show l) => Label l where
    canFlowTo :: l -> l -> Bool
    lub :: l -> l -> l -- Least upper bound
    glb :: l -> l -> l -- Greatest lower bound
```

Example label:

```
instance Label Integer where
  x `canFlowTo` y = x <= y
  lub = max
  glb = min
```

DCLabels

Disjunction Category Label

```
("amit" \/  "deian") %% ("amit")
```

- Labels are split into *secrecy* (read) and *integrity* (write) components
- Each component is a boolean formula over principals in *Conjunctive Normal Form*
- *Principals* are just strings – i.e. usernames, network endpoints…

DCLabels

Labels form lattice:

- $\langle S_1 \mathbin{\%\%} I_1 \rangle \sqsubseteq \langle S_2 \mathbin{\%\%} I_2 \rangle$ iff
  - $S_2 \Longrightarrow S_1$, and
  - $I_1 \Longrightarrow I_2$ (note reversed order)

DCLabels

Some noteworthy points on the lattice

- Top: nobody can read, everyone can write
  - `False %% True`
- Bottom: everybody can read, nobody can write
  - `True %% False`
- Public: everybody can read *and* write
  - `True %% True`

LIO - Labeled I/O

We saw it two slides ago… *canFlowTo*

- A Haskell Monad to replace the IO monad
  - Get to interpose on the “>>=” (bind) operator
- Every thread of execution has a “current label”
- Restricts code from performing unchecked side-effects (I/O, variable mutation)

LIO - Labeled I/O

Inputs, outputs, mutable variables, locks... are all labeled, so the TCB performs label checks:

```
hPutStr :: Labeled Handle -> String -> LIO ()
hPutStr (LabeledTCB hLabel h) str = do
  cl <- currentLabel
  if cl `canFlowTo` hLabel &&
       hLabel `canFlowTo` cl then
    -- raises current label to the glub of cl and h
    taint hLabel
    ioTCB $ hPutStr h
    else throwLIO LabelError {...}
```

LIO - Privileges

Sometimes we circumvent policies, but should be allowed if a thread is explicitly allowed to leak information.

- Privileges allow us to downgrade labels

```
class PrivDesc l p where
  canFlowToP :: p -> l -> l -> Bool
```

- $\langle S_1 \,\%\%\, l_1 \rangle \sqsubseteq_p \langle S_2 \,\%\%\, l_2 \rangle$ iff
  - $S_2 \wedge p \Longrightarrow S_1$, and
  - $l_1 \wedge p \Longrightarrow l_2$ (note reversed order)

Overflow

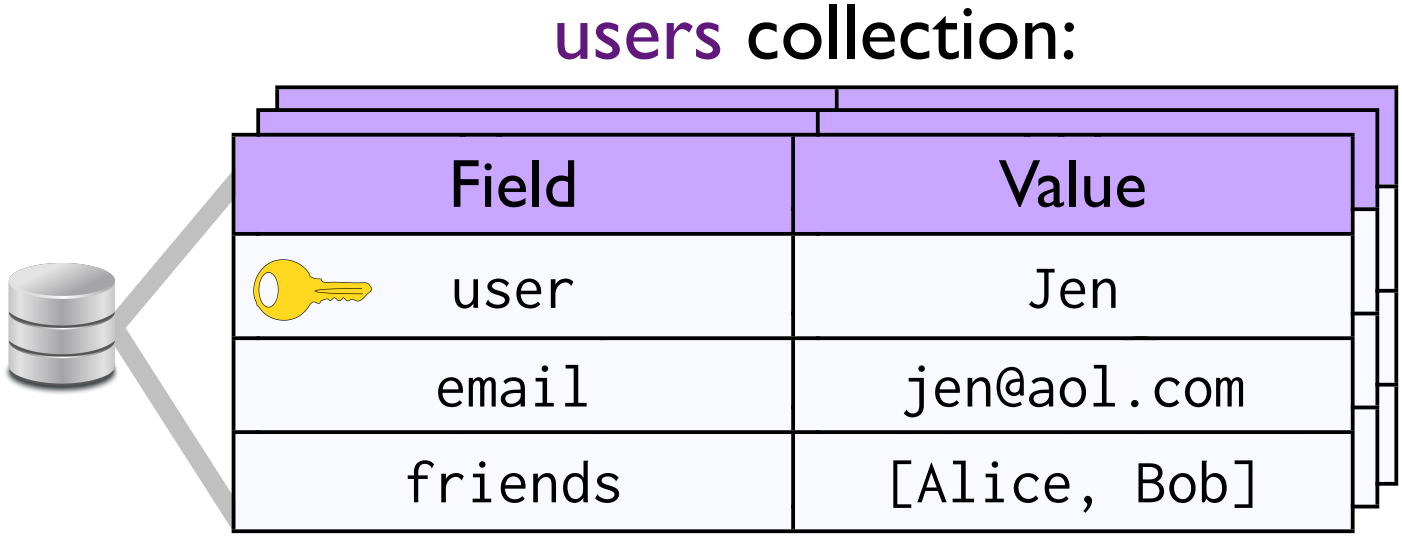
MPs and VCs

A closer look…

Model Policy

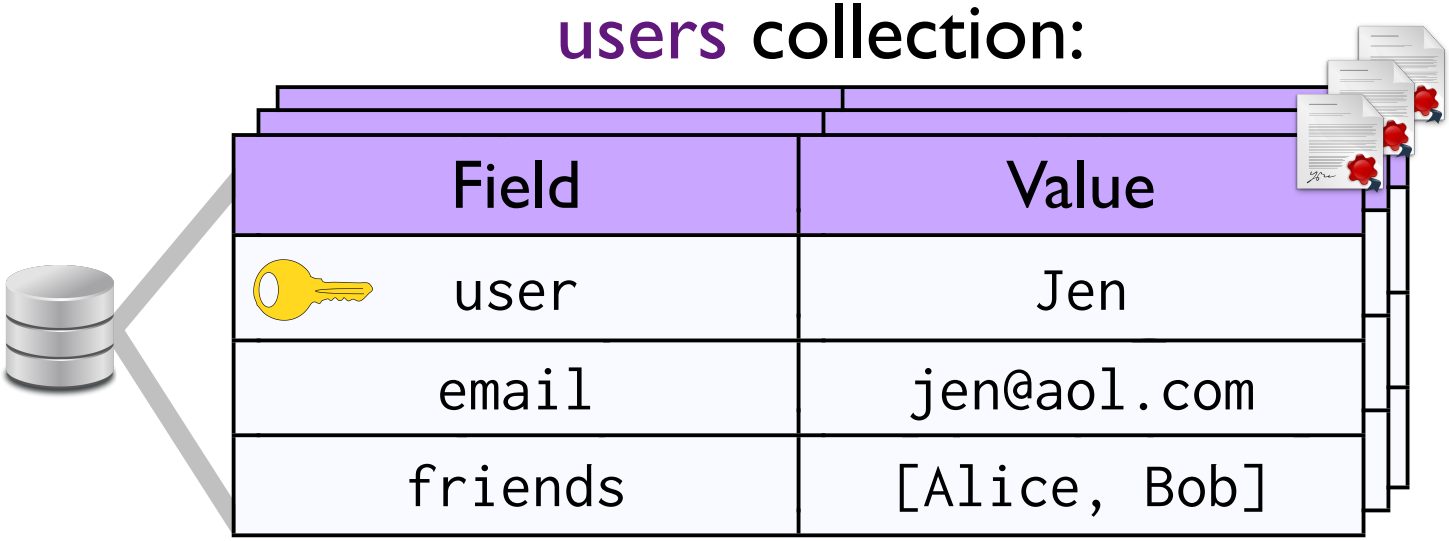
A document oriented data-store:

- Documents are stored in collections, stored in databases
- Semi-structured schema with flexible data-types



Model Policy

- Web app data *already* encodes policy
- Function from a document to a policy



```
collection "users" $ do
    access $ do
      readers ==> anybody
      writers ==> anybody
    field "user" key
    document $ \doc -> do
      readers ==> anybody
      writers ==> ("user" `from` doc)
    field "email" $ labeled $ \doc -> do
      readers ==> ("user" `from` doc)
              \/ fromList ("friends" `from` doc)
      writers ==> anybody
```

View Controller

- A VC is a web request handler
- Implement UI and external API
  - Source code viewer, RSS feed, Wiki editor,...
- Handle all data persistence through MPs
- Low barrier, since new VCs can reuse existing MPs

Bugs in VCs are manifested as broken features – never as vulnerabilities

Evaluation: Usablility

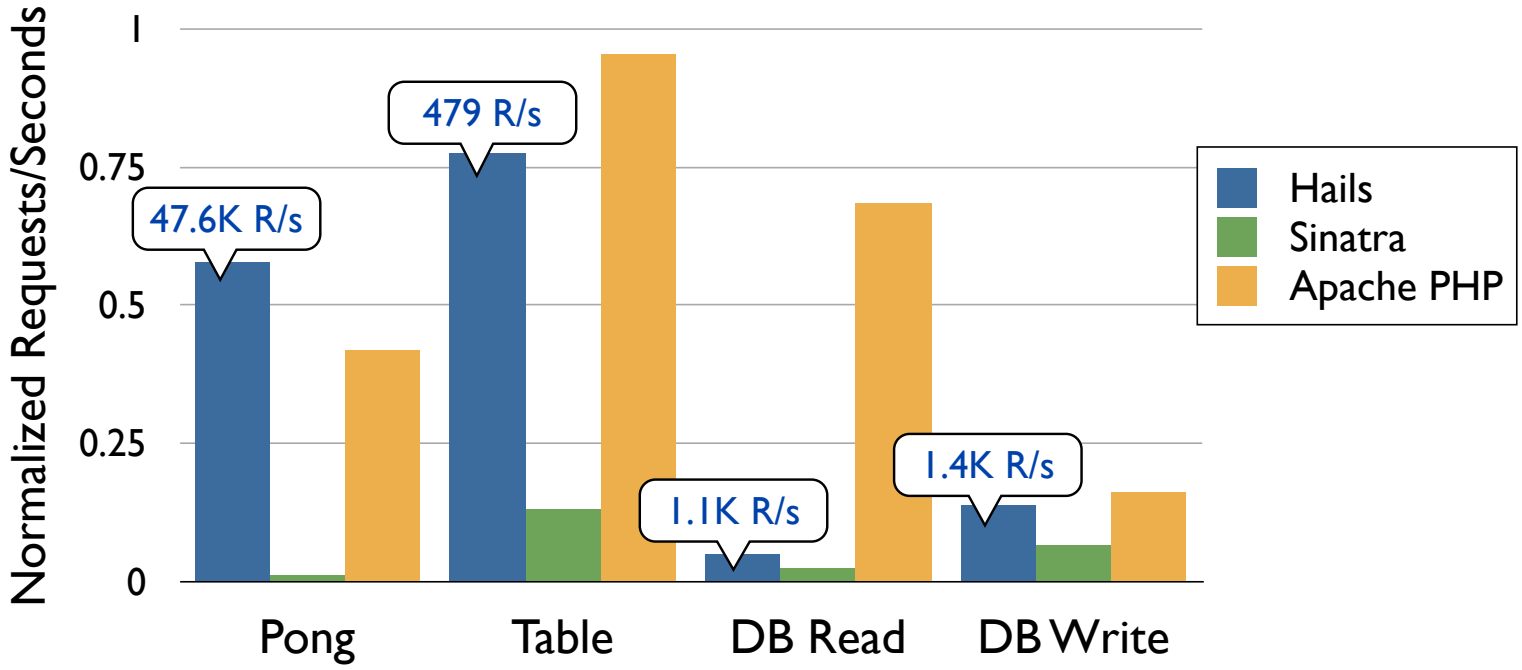
$\surd$ MPVC simplified reasoning about security $\surd$ Hails rendered common security bugs futile

χ Need scaffolding tools

χ ~~Writing policies is hard~~.

? Better with new policy DSL

Evalutaion: Performance



Limitations / Present & Future Work

- Confined to Haskell
  - Now - cjail
  - Future - Dune
- Covert channels
  - Internal timing closed ([ICFP 2012])
  - External timining - mitigation
  - How much to mitigate?
  - More work to do...
  - Cache-based timing attack

tl;dr

- Current platforms: functionality vs. privacy
- Hails platforms guarantee security end-to-end
  - Host apps on platform
  - Make policy explicit
  - Enforce policy with information flow control

```
$ cabal install hails
```

`http://gitstar.com` `http://hails.io/`