Hails: Protecting Data Privacy in Untrusted Web Applications

Daniel B. Giffin, Amit Levy, Deian Stefan, David Terei, John Mitchell, David Mazières, and Alejandro Russo
Web platforms are great!
They allow third-party developers to build apps that use our personal data.
Web platforms are **scary**! They allow third-party developers to build apps that use our personal data.
Trust concerns

• Don’t know the developers
  ➢ Cannot determine trustworthiness of apps

• They may be malicious or security-unaware

• Building secure web apps is hard
  ➢ Even well-meaning authors cannot be trusted
Typical App Design

Use popular MVC paradigm

**Model:** interface to data

**View:** renders pages

**Controller:** handles and responds to HTTP requests
How is security policy specified and enforced?

➤ E.g., only Jen’s friends may see her email address
Typical App Design

How is security policy specified and enforced?

- E.g., only Jen’s friends may see her email address

Intertwined throughout code

Error prone and not scalable
Can decide to give an app access to data, but can’t control how the app uses your data.
Is there any hope for privacy on platforms?
Change the hosting model

• Current model
  ➤ App developers host their own apps
  ➤ Platform enforces security: terms of service

• New model
  ➤ Platform provider hosts apps
  ➤ Platform enforces security: information flow control
Hails: A web platform framework

• Security policy is explicit and first-class
  ➤ Specified as single concise module

• Users still trust core platform components

• Apps are untrusted
  ➤ Language-level information flow control guarantees apps always obey policy
What makes Hails different?

Aeolus, HiStar, Nexus, Jif, Ur/Web, ...

• No guide for structuring applications
• Policies are hard to write
• Not appropriate for dynamic systems, e.g., web
• Modify entire application stack
Goals

• Deployable today!

• Usable by web developers

• Suitable for building extensible web platforms
  ➢ Enforcing policy across untrusted apps
Adding Policy to MVC

• New paradigm: Model-Policy-View-Controller
• Policy specified alongside data model
• No policy code in View or Controller
Two categories of code

Models-Policies (MPs)

- Specify data model and policy on data
- Users trust MPs they use to handle data

Views-Controllers (VCs)

- Implement UI and other functionality
- Users need not trust VCs

Policy enforced globally
Information flow control

• Policy specifies where data can flow
  ➢ **Wrong:** app can’t read Jen’s email address because it may leak it to Eve
  ➢ **Right:** app can read Jen’s email, but only reveal it to Jen, Alice or Bob

• Policy follows data through system

• Runtime enforces policy end-to-end
  ➢ E.g., when making HTTP request
Case study: GitStar

Hails

Haskell Web Platform Framework.

Repo: git clone ssh://deian@gitstar.com/scs/hails.git

- name
  - Hails
  - examples
  - tests
  - .gitignore
  - LICENSE
  - README.md
  - Setup.hs

size
--
--
--
5.0e-2 KB
0.7 KB
0.36 KB
5.0e-2 KB
Case study: GitStar

- GitStar provides
  - MPs that specify projects and users
  - VC for managing projects and users
- Third-party authors provide
  - Code viewer
  - Wiki
  - Follower app
  - etc.
Model-Policy

+ View-Controller
Model-Policy (MP)

Data model: document-oriented

- **Collection**: set of documents
- **Document**: set of field-value pairs

```
<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>Jen</td>
</tr>
<tr>
<td>email</td>
<td><a href="mailto:jen@aol.com">jen@aol.com</a></td>
</tr>
<tr>
<td>friends</td>
<td>[Alice, Bob]</td>
</tr>
</tbody>
</table>
```
Model-Policy (MP)

Data model: document-oriented

- Collection: set of documents
- Document: set of field-value pairs

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>Jen</td>
</tr>
<tr>
<td>email</td>
<td><a href="mailto:jen@aol.com">jen@aol.com</a></td>
</tr>
<tr>
<td>friends</td>
<td>[Alice, Bob]</td>
</tr>
</tbody>
</table>
Model-Policy (MP)

- Policy specifies restrictions on:
  - Collections, documents, fields
  - E.g., only Jen may modify her profile
  - E.g., only Jen and her friends may read her email address

- Policy composes
  - E.g., to read document you must be able to read the collection
Example: Enforcing policy

- **MP:**

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>Jen</td>
</tr>
<tr>
<td>email</td>
<td><a href="mailto:jen@aol.com">jen@aol.com</a></td>
</tr>
<tr>
<td>friends</td>
<td>[Alice, Bob]</td>
</tr>
</tbody>
</table>

  **Policy:** Only Jen, Alice and Bob can read

- **Eve’s untrusted address book VC:**
Example: Enforcing policy
Example: Enforcing policy

Bob's Browser -> AddrBook

Eve's spam server

GitStar User MP
Example: Enforcing policy

Bob’s Browser

AddrBook

Eve’s spam server

Eve’s server

Allow?

GitStar User MP +
Example: Enforcing policy
Example: Enforcing policy

Bob's Browser

AddrBook

findEmail users "user" "Jen"

Eve's spam server

Eve's server

GitStar User MP

Allow?
Example: Enforcing policy

Bob's Browser

AddrBook

findEmail users "user" "Jen"

GitStar User MP

Eve's spam server

Eve's server
Example: Enforcing policy

```
findEmail users "user" "Jen"
```

Policy: Only Jen, Alice and Bob can read

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>Jen</td>
</tr>
<tr>
<td>email</td>
<td><a href="mailto:jen@aol.com">jen@aol.com</a></td>
</tr>
<tr>
<td>friends</td>
<td>[Alice, Bob]</td>
</tr>
</tbody>
</table>
Example: Enforcing policy

Bob’s Browser -> AddrBook

findEmail users “user” “Jen”

jen@aol.com

Policy: Only Jen, Alice and Bob can read Eve’s server

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>Jen</td>
</tr>
<tr>
<td>email</td>
<td><a href="mailto:jen@aol.com">jen@aol.com</a></td>
</tr>
<tr>
<td>friends</td>
<td>[Alice, Bob]</td>
</tr>
</tbody>
</table>
Example: Enforcing policy

Policy: Only Jen, Alice and Bob can read

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>Jen</td>
</tr>
<tr>
<td>email</td>
<td><a href="mailto:jen@aol.com">jen@aol.com</a></td>
</tr>
<tr>
<td>friends</td>
<td>[Alice, Bob]</td>
</tr>
</tbody>
</table>
Example: Enforcing policy

/gitstar user MP

---

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>Jen</td>
</tr>
<tr>
<td>email</td>
<td><a href="mailto:jen@aol.com">jen@aol.com</a></td>
</tr>
<tr>
<td>friends</td>
<td>[Alice, Bob]</td>
</tr>
</tbody>
</table>

---

findEmail users "user" "Jen"

---

addrbook

---

Policy: Only Jen, Alice and Bob can read

---

Eve's spam server

---

Bob's Browser

---

Eve's server
Example: Enforcing policy

Policy: Only Jen, Alice and Bob can read Eve's server

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>Jen</td>
</tr>
<tr>
<td>email</td>
<td><a href="mailto:jen@aol.com">jen@aol.com</a></td>
</tr>
<tr>
<td>friends</td>
<td>[Alice, Bob]</td>
</tr>
</tbody>
</table>
Example: Enforcing policy

Bob's Browser

AddrBook

findEmail users "user" "Jen"

jen@aol.com

Eve's spam server

Eve's server

Policy: Only Jen, Alice, and Bob can read

GitStar User MP

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>Jen</td>
</tr>
<tr>
<td>email</td>
<td><a href="mailto:jen@aol.com">jen@aol.com</a></td>
</tr>
<tr>
<td>friends</td>
<td>[Alice, Bob]</td>
</tr>
</tbody>
</table>
Web app data models already encode policy

- Ownership
- Relationships between users
- …

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>Jen</td>
</tr>
<tr>
<td>email</td>
<td><a href="mailto:jen@aol.com">jen@aol.com</a></td>
</tr>
<tr>
<td>friends</td>
<td>[Alice, Bob]</td>
</tr>
</tbody>
</table>
Example: Policy specification

collection "users" $ do
  access $ do
    readers ==> anybody
    writers ==> anybody
  field "user" key

document $ \lambda doc \rightarrow do
  readers ==> anybody
  writers ==> ("user" `from` `doc`)

field "email" $ labeled $ \lambda doc \rightarrow do
  readers ==> ("user" `from` `doc`)
  writers ==> ("user" `from` `doc`)
  \lor fromList ("friends" `from` `doc`)
  writers ==> anybody
Example: Policy specification

collection “users” $ do
  access $ do
    readers ==> anybody
    writers ==> anybody

field “user” key

document $ λdoc -> do
  readers ==> anybody
  writers ==> ("user" `from` doc)

field “email” $ labeled $ λdoc -> do
  readers ==> ("user" `from` doc)
  writers ==> anybody

Collection is public
Example: Policy specification

collection “users” $ do
  access $ do
    readers ==> anybody
    writers ==> anybody

field “user” key

document $ \lambda doc -> do
  readers ==> anybody
  writers ==> (“user” `from` doc)

field “email” $ labeled $ \lambda doc -> do
  readers ==> (“user” `from` `doc`
    \lor fromList (“friends” `from` `doc`)
  writers ==> anybody
Example: Policy specification

collection “users” $ do
  access $ do
    readers ==> anybody
    writers ==> anybody

field “user” key

document $ \lambda doc -> do
  readers ==> anybody
  writers ==> (“user” `from` doc)

field “email” $ labeled $ \lambda doc -> do
  readers ==> (“user” `from` doc)
  writers ==> anybody

Index documents by user names
Example: Policy specification

collection “users” $ do
  access $ do
    readers ==> anybody
    writers ==> anybody
  field “user” key

document $ \lambda doc -> do
  readers ==> anybody
  writers ==> (“user” `from` doc)

field “email” $ labeled $ \lambda doc -> do
  readers ==> (“user” `from` doc)
  writers ==> everybody

  \lor fromList (“friends” `from` doc)
Example: Policy specification

collection “users” $ do
  access $ do
    readers ==> anybody
    writers ==> anybody

field “user” key

  document $ λdoc -> do
    readers ==> anybody
    writers ==> (“user” `from` doc)

field “email” $ labeled $ λdoc -> do
  readers ==> (“user” `from` doc)
  writers ==> anybody

Only Jen can modify document fields
Example: Policy specification

collection "users" $ do
  access $ do
    readers ==> anybody
    writers ==> anybody
  field "user" key

document $ \lambda doc -> do
  readers ==> anybody
  writers ==> ("user" `from` doc)
  field "email" $ labeled $ \lambda doc -> do
  readers ==> ("user" `from` doc)
    \lor fromList ("friends" `from` doc)
  writers ==> anybody
Example: Policy specification

collection “users” $ do
  access $ do
    readers ==> anybody
    writers ==> anybody
  field “user” key
document $ λdoc -> do
    readers ==> anybody
    writers ==> (“user” `from` doc)
  field “email” $ labeled $ λdoc -> do
    readers ==> (“user” `from` doc)
    writers ==> anybody
Example: Policy specification

collection “users” $ do
  access $ do
    readers ==> anybody
    writers ==> anybody
  field “user” key

document $ \lambda doc -> do
  readers ==> anybody
  writers ==> ("user" `from` doc)

field “email” $ labeled $ \lambda doc -> do
  readers ==> ("user" `from` doc)
  writers ==> anybody

  \lor fromList ("friends" `from` doc)
Model-Policy

View-Controller
View-Controller (VC)

- A VC is a request handler
- Provide application functionality
  - E.g., source code browser, blog editor, ...
- Invoke MPs to store/fetch user data
- Bugs in VCs are never vulnerabilities
  - Runtime enforces security policy
Model-Policy + View-Controller
Implications of MPVC

- Users: choose VCs based on functionality
- Devs: build apps on top of existing user-data
  - Models and policies are reusable
Implementation

• Hails is a Haskell library
  ➢ Quick turnaround on API design
  ➢ Developers can use existing tools and libraries

• Hails runtime system
  ➢ Provides HTTP server that invokes VC
  ➢ Enforces information flow at the language-level
Evaluation: Usability

✓ MPVC simplifies reasoning about security when building a platform

✓ Hails renders common security bugs futile  
  E.g., mass assignment vulnerability

✗ Need scaffolding tools

✗ Writing raw policy is hard

✓ Writing policy with DSL is simpler
Performance evaluation

<table>
<thead>
<tr>
<th></th>
<th>Hails</th>
<th>Sinatra</th>
<th>Apache PHP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pong</td>
<td>47.6K R/s</td>
<td>1.1K R/s</td>
<td>479 R/s</td>
</tr>
<tr>
<td>Table (DB Read)</td>
<td>479 R/s</td>
<td>1.4K R/s</td>
<td>1.1K R/s</td>
</tr>
<tr>
<td>DB Write</td>
<td>1.4K R/s</td>
<td>1.1K R/s</td>
<td>479 R/s</td>
</tr>
</tbody>
</table>
Conclusions

• Current platforms: functionality vs. privacy

• Hails platforms guarantee security across apps
  ➤ Host apps on platform
  ➤ Make policy explicit
  ➤ Enforce policy with information flow control

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

$ cabal install hails