Flogent

Information Flow Control Security Feature for Cogent

Vivian Dang (UNSW)
Supervisor: Christine Rizkallah

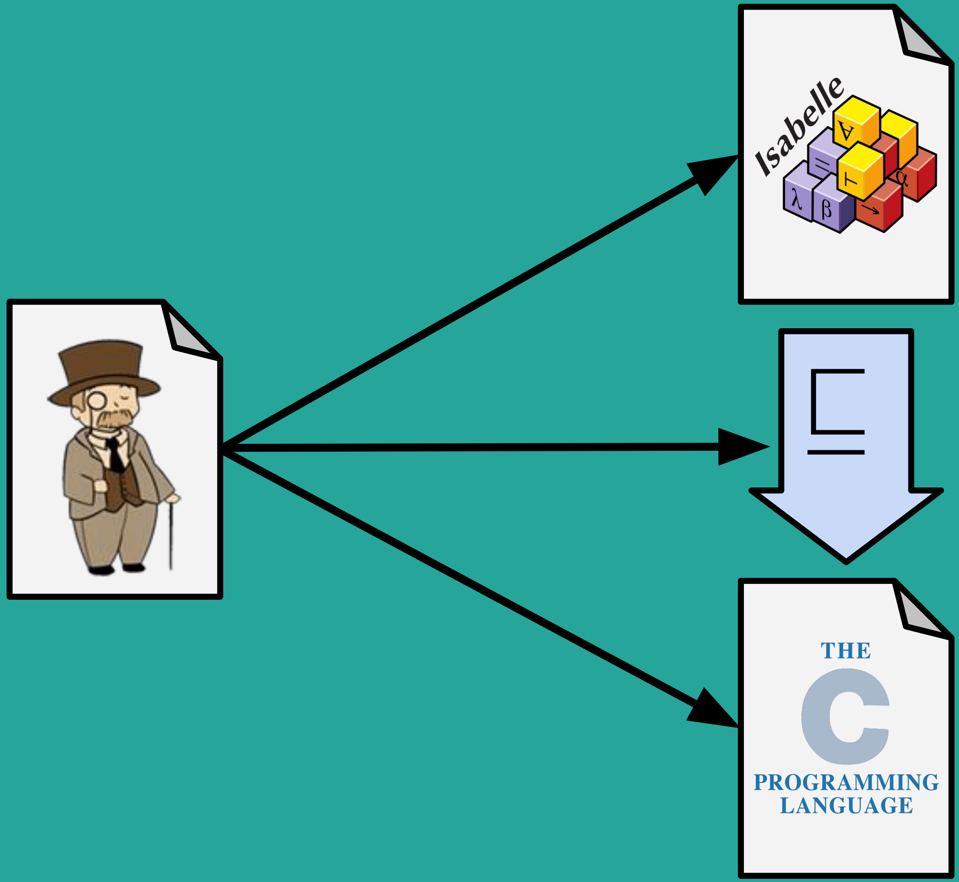
@viv_yd

Information Flow Security Feature on Cogent

Cogent

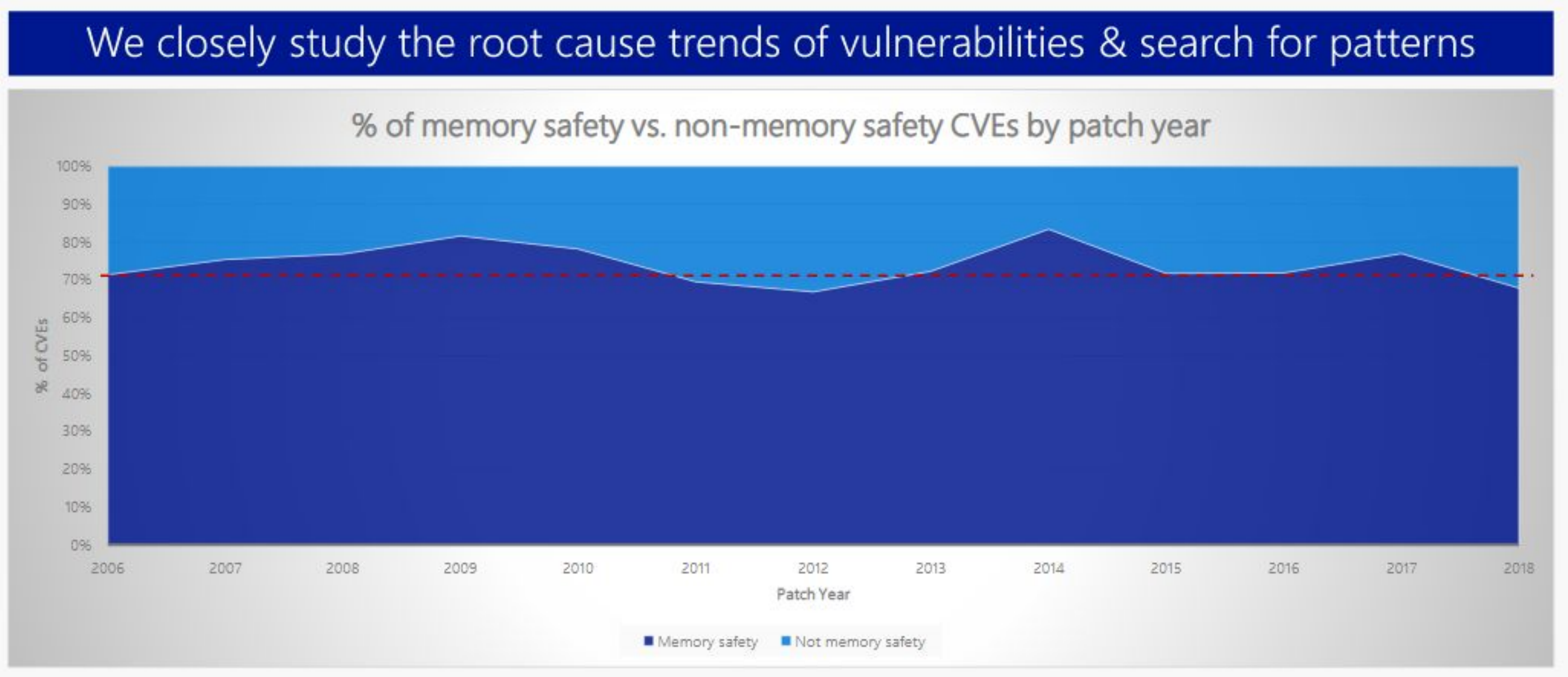
Cogent

- Purely Functional Language
    - Linear Type System (Uniqueness type)
- Easy to reason semantics
- Aim to reduce cost of verification


Isabelle
⊑
THE
C
PROGRAMMING LANGUAGE

The “Why”

Causes of Security Vulnerabilities



We closely study the root cause trends of vulnerabilities & search for patterns

% of memory safety vs. non-memory safety CVEs by patch year

Matt Miller, Presentation at BlueHat 2019, Microsoft

Memory Safety

- Temporal Safety
  - use of uninitialised memory, use-after-free
  - handled by Linear Types in Cogent ✓
- Spatial Safety
  - buffer overflow/array out-of-bounds
  - stack smashing
  - handled by Type Safety in Cogent ✓

What about the other 30%?

- Hardware problems:
  - Spectre and Meltdown

- Information Flow Vulnerabilities
  - A growing source of security issues *

* Stephan Neuhaus and Thomas Zimmermann, *Security Trend Analysis with CVE Topic Models*, ISSRE 2010, IEEE.

Information Flow Vulnerabilities:

- Secret data is accessed or modified by insecure processes
- Breach of Confidentiality or Integrity

- Examples:
  - Password leaks
  - Side channel attack: Timing, Power Analysis
  - Injections

My focus: vulnerabilities observable in the language semantics

The “What”

Flogent

Information Flow Control (IFC)

- Information is tagged by different security levels
- Arranged in a lattice or order
- Info can only flow from low to high

Work of Abadi et al.

- Seminal work for security types (420 citations)

- Monads to type computations with their levels
- Tags are part of the type system (static semantics)
- No run-time tagging -> No performance impact
- Checking at compile-time -> Early feedback

MAC (Mandatory Access Control)

- Haskell implementation of Abadi’s work
- Statically-enforced IFC library for Haskell

M. Vassena et al. (2017)

MAC - Overview

- Computations are labelled with security level
  - must have authorisation to perform
- Information is labelled
  - must have authorisation to read

M. Vassena et al. (2017)

MAC - Overview

MAC library has two main operations:

- unlabel
    - allow un-labelling information if authorisation is satisfied
- join
    - transform computations from high-level security to low-level security
    - but observing result is stopped by labelling

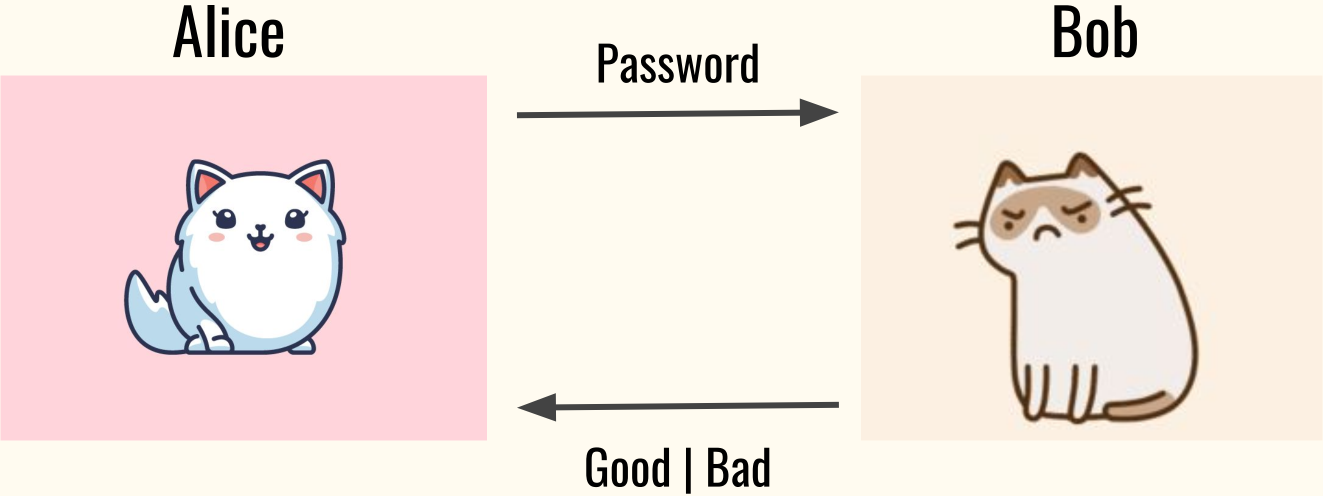
MAC Operations

$$unlabel :: \ell_L \sqsubseteq \ell_H \Rightarrow Labeled\ \ell_L\ a \rightarrow MAC\ \ell_H\ a$$

$$join :: \ell_L \ \sqsubseteq \ \ell_H \Rightarrow MAC\, \ell_H\ \tau \rightarrow MAC\, \ell_L\ (Labeled\, \ell_H\ \tau)$$

Scenario

Case study



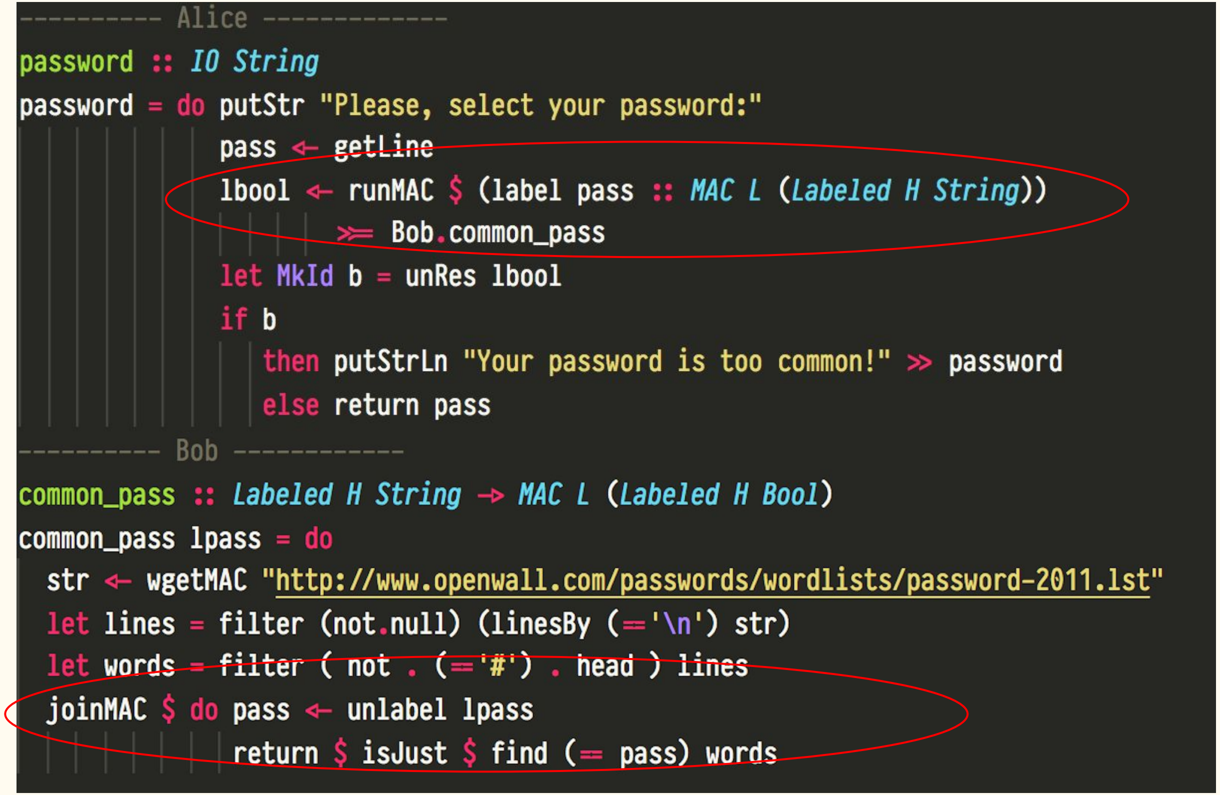
Alice

Password

Bob

Good | Bad

MAC Library



```
---------- Alice -------------
password :: IO String
password = do putStr "Please, select your password:"
              pass <- getLine
              lbool <- runMAC $ (label pass :: MAC L (Labeled H String))
                      >>= Bob.common_pass
              let MkId b = unRes lbool
              if b
                 then putStrLn "Your password is too common!" >> password
                 else return pass
---------- Bob ------------
common_pass :: Labeled H String -> MAC L (Labeled H Bool)
common_pass lpass = do
  str <- wgetMAC "http://www.openwall.com/passwords/wordlists/password-2011.lst"
  let lines = filter (not.null) (linesBy (=='\n') str)
  let words = filter ( not . (=='#') . head ) lines
  joinMAC $ do pass <- unlabel lpass
               return $ isJust $ find (== pass) words
```

Relevance to Cogent

- Purely functional language
- Static approach - good for systems programming

Differences

- Cogent doesn’t support Monads
- Cogent has Linear Types
  - variables of this type must be used exactly once

Cogent doesn’t support monads

Haskell $wget :: URL \to IO\ String$

Cogent $wget :: (🌍, URL) \to (🌍, String)$

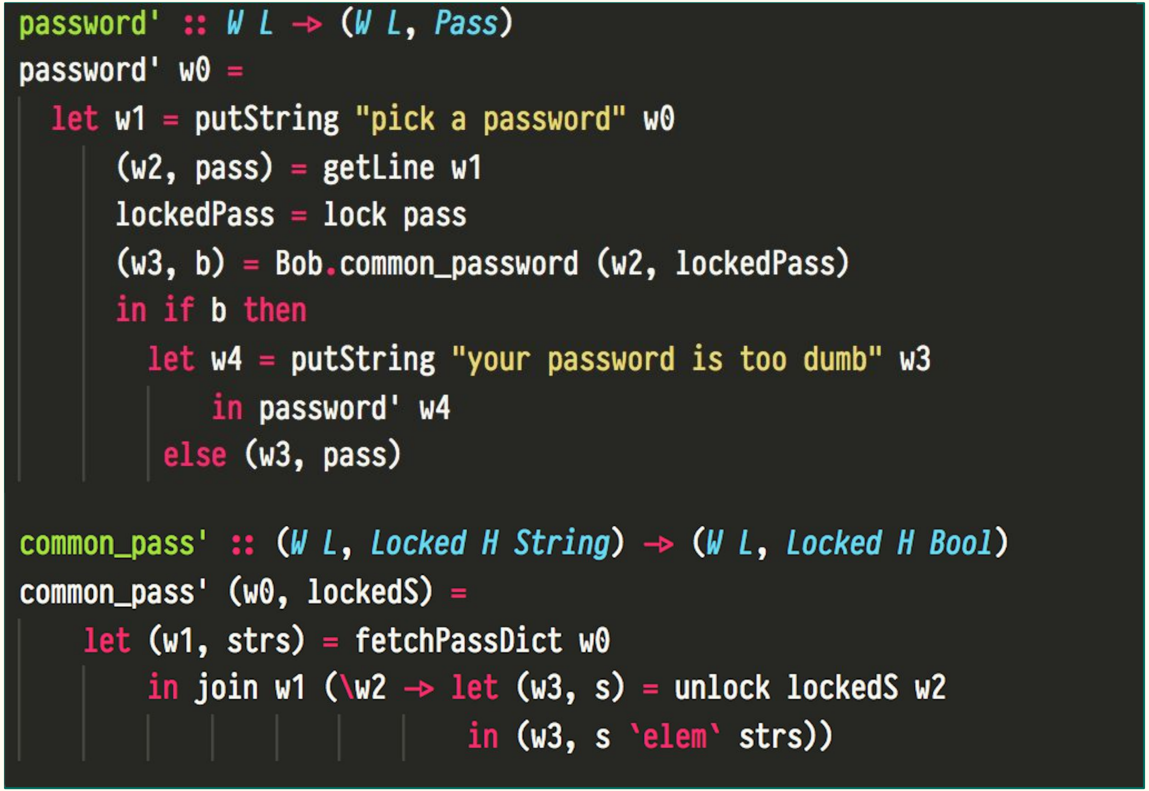
Flogent

- Tagging the World with a security level
- Utilise the Linearity of the types

$$
\begin{aligned}
join :: \ell_L \sqsubseteq \ell_H \ \Rightarrow&\ \text{🌍}_{\ell_L} \rightarrow (\text{🌍}_{\ell_H} \rightarrow (\text{🌍}_{\ell_H}, \tau)) \\
\rightarrow&\ (\text{🌍}_{\ell_L}, Locked\ \ell_H\ \tau)
\end{aligned}
$$

$$
unlock :: \ell_L \sqsubseteq \ell_H \Rightarrow Locked\ \ell_L\ \tau \rightarrow \text{🌍}_{\ell_H} \rightarrow (\text{🌍}_{\ell_H}, \tau)
$$

Flogent



```
password' :: W L -> (W L, Pass)
password' w0 =
  let w1 = putString "pick a password" w0
      (w2, pass) = getLine w1
      lockedPass = lock pass
      (w3, b) = Bob.common_password (w2, lockedPass)
      in if b then
        let w4 = putString "your password is too dumb" w3
            in password' w4
         else (w3, pass)

common_pass' :: (W L, Locked H String) -> (W L, Locked H Bool)
common_pass' (w0, lockedS) =
    let (w1, strs) = fetchPassDict w0
        in join w1 (\w2 -> let (w3, s) = unlock lockedS w2
                            in (w3, s `elem` strs))
```

Flogent vs MAC

MAC	Flogent
unsafePerformIO	Generated C code can call to unsafe C functions
Exceptions	No Exceptions
Non-termination	All Cogent functions terminate

Current status

- Implemented the join and unlock operations in (mini) Cogent
- More testing WIP

What’s Next?

Future work

- Some more case studies to test expressiveness
- Formalisation
- Making sure that security properties (confidentiality & integrity) hold

Thank you!

@viv_yd

Citations

1. M. Vassena, A. Russo, P. Buiras, L. Waye, (2017) ‘Mac - A verified static information-flow control library’, Journal of Logical and Algebraic Methods in Programming, Elsevier.
2. Martín Abadi, Anindya Banerjee, Nevin Heintze, and Jon G. Riecke (1999). “A Core Calculus of Dependency”. Proceedings of POPL ’99. ACM.
3. A. Russo, (2015) ‘Two Can Keep a Secret, If One of Them Uses Haskell’, Proceedings of ICFP 2015, ACM.
4. L. O'Connor et al. (2016) ‘Refinement Through Restraint: bringing down the cost of verification’, Proceedings of ICFP 2016, ACM.