SAFE HASKELL

David Terei
David Mazières

Stanford University

Simon Marlow
Simon Peyton Jones

Microsoft Research
Haskell is a great language for building secure systems in:

- Information flow control
- Capabilities
- Computations on encrypted data

But all this work can’t secure untrusted code in the real world!
MOTIVATION

Running Example:

Build in Haskell a website that can run untrusted third-party plugins:

- Users can upload plugins in source form
- Any user can install an uploaded plugin against their account
MOTIVATION

How?

• Carefully craft plugin interface to restrict functions that a plugin can execute
  • e.g Only pure functions

• Need type safety guarantees for this to work
MOTIVATION

\[ f :: a \to a \]
MOTIVATION

\[ f :: a \rightarrow a \]

\[
f a = \text{unsafePerformIO} \$ \ do \\
_ <- \text{send_credit_card} \\
\text{return} \ a
\]
SOLUTION?
SOLUTION?

Safe Haskell!
SAFE HASKELL

• Safe subset of Haskell that provides ‘enough’ guarantees

• A safe import extension

• A definition of trust that applies to modules and packages
SAFE LANGUAGE

• Safe language (enabled with -XSafe) provides:
  
  • Type safety
  
  • Guaranteed module boundaries
  
  • Semantic consistency
  
• These are the properties that we usually think about Haskell having

• Safe language is a subset of Haskell
-XSAFE RESTRICTIONS

- FFI imports must be in the IO monad
- Can’t define RULES
- No Template Haskell
- No GeneralizedNewtypeDeriving
- No hand crafted instances of Data.Typeable, only derived
- Overlapping instances can only overlap instances defined in the same module
- Can only import other ‘trusted’ modules
So for untrusted plugins, compile with 

-\texttt{XSafe}

Can craft a plugin interface that uses types carefully to control functions a plugin can execute

Done?
TURTLES ALL THE WAY DOWN

- **-XSafe** compiled modules can only import trusted modules.

- So far **-XSafe** is only way to create trusted modules.

- What about modules like **Data.ByteString**?
  - Want to allow untrusted code to use **Data.ByteString**
  - Unsafe internals but safe API
-XTRUSTWORTHY

Allows a module author to declare:

‘While module M may use unsafe functions internally, it only exposes a safe API’
-XTRUSTWORTHY

- No restrictions on Haskell language
- Marks a module as trusted though
- Module author should assure that type safety can’t be violated by importing their module
- Enables a small extension called safe imports
WHAT IS TRUST?

• What determines if a module is considered ‘trusted’?
  
  • `-XSafe` compiled modules
  
  • What about `-XTrustworthy` modules?
WHAT IS TRUST?

• -XTrustworthy allows a module author to mark any module as potentially ‘trusted’

• Very easy to abuse

• So we require that the client (person running the compiler) assert that they trust the module author by stating they trust the package

• For example:

  • Don Stewart marks Data.ByteString as Trustworthy

  • Untrusted plugin author imports and uses Data.ByteString

  • Website administrator marks the bytestring package as trusted
WHAT IS TRUST?

• For \texttt{-XSafe}:
  • trust provided by \texttt{compiler}

• For \texttt{-XTrustworthy}:
  • trust of module stated by \texttt{module author}
    • trust of module author provided by \texttt{client} by trusting the package the module resides in
TRUST IS TRANSITIVE

• For A to be trusted package P must be trusted
• An -XSafe module may bring in a package trust requirement

{-# LANGUAGE Safe #-}
module A
...

Package P

{-# LANGUAGE Trustworthy #-}
module B
...

{-# LANGUAGE Safe #-}
module A
...

An -XSafe module may bring in a package trust requirement
PACKAGE TRUST

• ghc-pkg trust <pkg>
• ghc-pkg distrust <pkg>
• ghc -trust <pkg> ...
• ghc -distrust <pkg> ...
• ghc -distrust-all-packages ...
SAFE IMPORTS

• One extension to the Haskell language:

import safe M

• Allows module author to specify that M must be trusted for the import to succeed

• Under -XSafe all imports are safe imports (keyword implicit)

• Under -XTrustworthy the module author can choose
PROBLEMS WITH 7.2

• Current description is of Safe Haskell in 7.2

• Issue with operation of package trust

  • Causes Safe Haskell to be invasive, infect the world!
"I'm running into a lot of issues like the following:

libraries/hoopl/src/Compiler/Hoopl/Collections.hs:14:1:

    base:Data.List can't be safely imported! The package (base) the module resides in isn't trusted."
PACKAGE TRUST REIFIED

• In 7.4, we won’t require that the package a Trustworthy module resides in be trusted for the compilation to succeed

• -XTrustworthy modules will simply be trusted by default

• New -fpackage-trust flag to enable old behavior of 7.2

  • This flag should always be used if you are compiling untrusted code
SAFE INference

Unreasonable to expect the Haskell world to all start putting explicit `-XSafe` and `-XTrustworthy` pragmas in their files.

So in 7.4:

- Safe status of a module will be inferred
- New `-XUnsafe` flag to explicitly mark a module as unsafe so that it can’t be imported by untrusted code
module RIO.Unsafe ( RIO(..) ) where

newtype RIO a = UnsafeRIO { runRIO :: IO a }

instance Monad RIO where
    return = UnsafeRIO . return
    (UnsafeRIO m) >>= k = UnsafeRIO $ m >>= runRIO . k

module RIO.FileAccess ( rioReadFile, rioWriteFile ) where

... pathOK f = {- Implement some policy -}

rioReadFile :: FilePath -> RIO String
rioReadFile f = UnsafeRIO $ do
    ok <- pathOK f
    if ok then readFile f else return ""

rioWriteFile :: FilePath -> String -> RIO ()
rioWriteFile f s = ...
RUNNING EXAMPLE

{-# LANGUAGE Trustworthy #-}

module RIO ( RIO() , runRIO, rioReadFile, rioWriteFile ) where

import RIO.Unsafe

import safe RIO.FileAccess

{-# LANGUAGE Safe #-}

module UntrustedPlugin ( runPlugin ) where

import RIO

runPlugin :: RIO ()
runPlugin = ...
SUMMARY

- New language flags: -XSafe, -XTrustworthy, -XUnsafe
- New option flag: -fpackage-trust (7.4)
- Safe status of a module will be inferred (7.4)

Trust your types!
FUTURE WORK

• Prove safety guarantees

• Establish clearer definition of safe and what guarantees trustworthy modules should provide
  • Machine checking possible here?

• Do a retake on Safe language but by starting with a small, proven correct core and expanding out.
  • Inclusion in the Safe language could be used as a quality bar for new Haskell extensions.
  • Require formal semantics and proofs
SAFE HASKELL

In GHC 7.2

Please try out and provide feedback

http://www.scs.stanford.edu/~davidt/safehaskell.html