Towards Reasoning in the presence of code of unknown provenance

- or, trust and risk in an open world -

Lorentz Workshop 2015

Sophia Drossopoulou (Imperial), James Noble (Victoria University Wellington), Mark S. Miller (Google), Toby Murray (NICTA)
Reasoning with Code of Unknown Provenance

Hoare Rules - Method Call

When receiver belongs to class \( C \)

\[
\begin{align*}
\text{PRE}(m,C) &= P \\
\text{POST}(m,C) &= Q \\
\text{x : C \land P[x/this,y/par]} & \{ z = x.m(y) \} \ Q[x/this,y/par,z/res]
\end{align*}
\]

What if I know nothing about receiver?

\[
\text{true } \{ z = x.m(y) \} \text{ true}
\]
Trust and Risk in Open Systems
- research questions -

• Objects collaborate with other objects of unknown provenance.

• Objects may unknowingly be dealing with malicious objects; they are therefore exposed to risks. Nevertheless, they proceed with the business.

• **No central trusted authority.**

• Therefore,
  
  • “our” code must be very “robust”,
  
  • we need means to specify trust and risk.
  
  • we need means to reason about adherence to such specifications.
Trust and Risk in Open Systems
- our contributions -

• To specify trust and risk, we propose
  • \textit{obeys} predicate: an object adheres to a specification,
  • \textit{MayAccess} predicate: an object may read some property
  • \textit{MayAffect} predicate: an object may affect some property
  • specifications talk of necessary conditions
• Predicates \textit{obeys}, \textit{MayAccess} and \textit{MayAffect} are hypothetical and often conditional.
• Hoare logic rules to reason about trust and risk.
• Apply our ideas on the Escrow Exchange (Miller et.al., ESOP’13).

\textbf{Simplifying Assumptions:}

• We do not consider concurrency and distribution (code in ESOP’13 does).
• We assume that different arguments to our methods are not aliases (but easy to expand).
• We do not consider whether two Accounts can trade (but \textit{CanTrade} in the PLAS paper).
Our findings for the Escrow

- We could write the specification.

- We could prove adherence to specification (by hand).

- The specification is weaker than we, and the Escrow authors, had expected. And no code can satisfy the stronger specification (we think).
Remaining Talk

Terminology: open world, trust and risk

Escrow Agent - Our running example

Hoare Logic
Terminology: open, trust and risk
What do we mean by open system?

We model open systems through dynamic linking of any, unknown, potentially malicious module \( M' \).

**Definition**

\[
M \models \text{Policy} \iff \\
\forall M'. \\
\forall \kappa \in \text{Arising}(M'*M): \ M'*M, \kappa \models \text{Policy}
\]

\( M' \) represents the “adversary”.

\( \text{Arising}(M'*M) \) restricts configurations to those reachable though execution of code from \( M'*M \).
What do we mean by trust?

Trust is relative to a configuration ($\kappa$), an object reference ($o$) and a policy-specification ($Policy$).

trust is hypothetical; no “trust” bit.

Definition

\[ M, \kappa \models o \text{ obeys Spec} \quad \text{iff} \]
\[ \forall \text{ Policy } \in \text{ Spec.} \]
\[ \forall \kappa' \in Reach(M, \kappa): \ M, \kappa' \models \text{Policy}[o/this] \]

\text{Reach}(M, \kappa): \text{ intermediate configurations reachable from } \kappa.
What do we mean by risk?

Risks are effects against which we want to guard our objects.

**policy** PolDeal_1:

**pre:** ....

{ res = this.deal(m,g) ; }

**post:** ....

∀ p. p obeys ValidPurse ....

[ p.balance = p.balance_{PRE} ∨
  ∃bp. bp=... ¬ (bp obeys ValidPurse) ∧ MayAccess_{PRE} (bp,p) ]
Escrow Agent - Our running example
Escrow Agent - Remit
(proposed by Miller, van Cutsem, Tulloh, ESOP 2013)

• Buyer and Seller want to exchange goods for money.

• Buyer does not trust Seller; Seller does not trust Buyer.

• Escrow Agent to make the exchange.

• If insufficient money or goods, then no exchange.

• Escrow Agent does not trust Buyer nor Seller, nor any Banks.

• Escrow Agent to mitigate risk to Buyer and Seller.
Escrow Agent - First Attempt
Exchange of \textbf{g} goods for \textbf{m} money

1. pay \textbf{m} to escrowMoney from buyerMoney
2. if no success then exit

// sufficient money
3. pay \textbf{g} to escrowGoods from sellerGoods

4. if no success then
   pay \textbf{m} to buyerMoney from escrowMoney
   exit

// sufficient money and goods
5. pay \textbf{g} to buyerGoods from escrowGoods

6. pay \textbf{m} to sellerMoney from escrowMoney
The Escrow purses

• intermediate store of \( m \) money, and \( g \) goods

• allow exchange to be undone, if insufficient goods or money

• Agent interrogates the escrow purses, to determine whether deposits were successful.

• Therefore, the correctness of process depends on the integrity of the escrow purses.

• But … where do escrow purses come from?
Where do Escrow Purses come from?

• The Agent has them before the transaction.

  **No!** This would require the Agent to know about all possible purses. Remember, no central authority.

• Seller and Buyer supply the escrows purses.

  **No!** It would require Seller and Buyer to have agreed before the transaction. Remember: Seller and Buyer do not trust each other.

• The Agent asks the associated Banks to supply the escrows purses.

  **No!** It would require the Agent to know about all possible banks. Remember, no central authority.

• The Agent asks `sellerMoney` to make one, and `buyerGoods` to make another one.

  **Yes!**
Exchange of $g$ goods for $m$ money

1a. $\text{escrowMoney} = \text{sellerMoney}.\text{sprout}()$
1b. $\text{res} = \text{escrowMoney}.\text{deposit}((\text{buyerMoney}, m))$

2. if !$\text{res}$ then exit

// sufficient money
3a. $\text{escrowGoods} = \text{buyerGoods}.\text{sprout}()$
3b. $\text{res} = \text{escrowGoods}.\text{deposit}((\text{buyerGoods}, g))$

4. if !$\text{res}$ then
   $\text{buyerMoney}.\text{deposit}((\text{escrowMoney}, m))$
   exit

// sufficient money and goods
5. $\text{buyerGoods}.\text{deposit}((\text{escrowGoods}, g))$

6. $\text{sellerMoney}.\text{deposit}((\text{escrowMoney}, m))$
Risk and Trust
Has Escrow Agent version1 fulfilled its remit?

• Buyer and Seller want to exchange goods for money.

• Buyer does not trust Seller; Seller does not trust Buyer.

• Escrow Agent to make the exchange.

• If insufficient money or goods, then no exchange.

• Escrow Agent does not trust Buyer nor Seller, nor any Banks.

• Escrow Agent to mitigate risk to Buyer and Seller.
1a. escrowMoney = sellerMoney.sprout()
1b. res = escrowMoney.
    deposit (buyerMoney, m)
2. if !res then exit

More Risks
• Similar risk scenarios exist for malicious escrowGoods.
• Similar risk scenarios exist for when escrowMoney is sprouted from buyerMoney, or when escrowGoods is sprouted from sellerMoney.
Escrow Agent - Second Attempt
• Extend Purse’s remit to ascertain trust and limit risk.

• Add introductory phase to Escrow Agent code, which assesses trustworthiness of Purses.
Escrow Agent - Second Attempt

ValidPurse specification
ValidPurse specification v2- overview

**specification** ValidPurse{

**policy** Pol_deposit_1:
  successful report, implies trust and enough funds, and transfer

**policy** Pol_deposit_2:
  unsuccessful report, implies no trust or not enough funds, and no transfer

**policy** Pol_sprout:
  sprout() creates a Purse of same trustworthiness

**policy** Pol_protect_balance:
  balance cannot be affected, unless you hold the purse itself
ValidPurse - deposit_1

successful report, implies trust and enough funds; transfer

policy Pol_deposit_1:

pre: amt : Number ∧ amt ≧ 0
{ res=this.deposit(prs, amt) }

post:
res = true →
// FUNCTIONAL
    prs.balance_pre - amt ≧ 0 ∧
    prs.balance = prs.balance_pre - amt ∧
    this.balance = this.balance_pre + amt ∧
// TRUST
   prs obeys ValidPurse ∧
// RISK

Note: conditional trust

[ MayAccess(o,p) → MayAccess_{pre}(o,p) ] )
ValidPurse - deposit_2

unsuccessful report, implies no trust or insufficient trust, and no transfer

policy Pol_deposit_2:

pre: amt : Number ∧ amt ≥ 0

{ res=this.deposit(prs, amt) }

post:
res = false →

// FUNCTIONAL & TRUST
prs.balance_{pre} - amt < 0 ∨ NOT( prs obeys ValidPurse)

// RISK
∀ p, o.
( p obeys_{pre} ValidPurse ∧ o :_{pre} Object. →
  p.balance_{pre} = p.balance ∧
  [ MayAccess(o,p) → MayAccess_{pre}(o,p) ]}
ValidPurse - sprout

sprout creates a Purse of same trustworthiness

**policy** Pol_sprout:

**pre:** true

{ res=this.spout() }

**post**

// FUNCTIONAL & TRUST
res obeys ValidPurse ∧ res.balance=0

// RISK
∀ p, o.
( p obeys\(_{\text{pre}}\) ValidPurse ∧ o :\(_{\text{pre}}\) Object. →
 ( p≠prs,this → p.balance\(_{\text{pre}}\) = p.balance ) ∧
 [ MayAccess(o,p.balance) → MayAccess\(_{\text{pre}}\)(o,p.balance) ] }
ValidPurse - protect_balance

balance cannot be affected, unless you hold the purse itself

**Policy** Pol_protect_balance:

\[ \forall p, o. \quad (p \text{ obeys ValidPurse} \land o : \text{Object.} \rightarrow [\text{MayAffect}(o,p.\text{balance}) \rightarrow \text{MayAccess}(o,p)] ) \]

**Note** - necessary, rather than sufficient condition
Escrow Agent - Second Attempt
code
EscrowAgent - establishing trust

escrowMoney = sellerMoney.sprout()
// sellerMoney obeys ValidPurse → escrowMoney obeys ValidPurse

res = escrowMoney.deposit (buyerMoney, 0)
// res=true ∧ escrowMoney obeys ValidPurse
// → buyerMoney obeys ValidPurse

if !res then exit
// sellerMoney obeys ValidPurse → ¬(buyerMoney obeys ValidPurse)

// sellerMoney obeys ValidPurse → buyerMoney obeys ValidPurse

res = buyerMoney.deposit (escrowMoney, 0)
// res=true ∧ buyerMoney obeys ValidPurse
// → escrowMoney obeys ValidPurse

if !res then exit

res = escrowMoney.deposit (buyerMoney, 0)
if !res then exit

// buyerMoney obeys ValidPurse ↔ seller obeys ValidPurse
EscrowAgent - the risk while establishing trust

```plaintext
escrowMoney = sellerMoney.sprout()

// ∀p. p obeys^{PRE} ValidPurse →
// [ p.balance^{PRE}=p.balance  ∨
// MayAccess^{PRE}(sellerMoney,p) ∧ ¬(sellerMoney obeys ValidPurse)
res= escrowMoney. deposit (buyerMoney,0)
// ....

if !res then exit   // ....

res= buyerMoney. deposit (escrowMoney,0)

//∀p. p obeys^{PRE} ValidPurse →
// [ p.balance^{PRE}=p.balance  ∨
// MayAccess^{PRE}(sellerMoney,p) ∧ ¬(sellerMoney obeys ValidPurse) ∨
// MayAccess^{PRE}(buyerMoney,p) ∧ ¬(buyerMoney obeys ValidPurse) ]

if !res then exit   // ....

res= escrowMoney. deposit (buyerMoney,0)
// ....

if !res then exit

//∀p. p obeys^{PRE} ValidPurse →
// [ p.balance^{PRE}=p.balance  ∨
// MayAccess^{PRE}(sellerMoney,p) ∧ ¬(sellerMoney obeys ValidPurse) ∨
// MayAccess^{PRE}(buyerMoney,p) ∧ ¬(buyerMoney obeys ValidPurse) ]

```

```
EscrowAgent  the full code

1st phase:
trustworthiness  buyerMoney and sellerMoney — as in previous slide

2nd phase:
trustworthiness buyerGood and sellerGood — similar to previous slide

3rd phase:
Do the transaction — as a couple of slides ago
Escrow Agent - Second Attempt

The specification
EscrowAgent specification - overview

Escrow Agent has fields buyerMoney, sellerMoney, buyerGoods, sellerGoods.

For the method call
this.deal(m,g)
we have four cases:

**policy** Pol_deal_1:
no malicious participants, sufficient money/goods, deal proceeds
res==true

**policy** Pol_deal_2:
no malicious participants, **insufficient** money/goods, deal does **not** proceed
res==false

**policy** Pol_deal_3:
buyer or seller malicious, deal does **not** proceed
res==false

**policy** Pol_deal_4:
buyer *and* seller malicious, deal **does** proceed
res==true
**Pol_deal_1**

no malicious participants, sufficient money/goods, deal proceeds

res==true

**policy Pol_deal_1:**

**pre:** m,g: Number

{ res = this.deal(m,g) ; }

**post:** res ∧ MalPartPurses = ∅

⇒

buyerMoney.balance_{PRE} ≧ m ∧ sellerMoney.balance_{PRE} ≧ g ∧

buyerMoney=buyerMoney_{PRE} - m ∧ sellerMoney=sellerMoney_{PRE} + m ∧

buyerGoods=buyerGoods_{PRE} + g ∧ sellerGoods=sellerGoods_{PRE} - g

∀ p. p ∈_{PRE} OtherPurses. p.balance = p.balance_{PRE} ∧

∀ o,p. o ∈_{PRE} Object. p obeys_{PRE} ValidPurse .

MayAccess(o,p.balance) ⇒ MayAccess(o,p.balance)_{PRE}

MalPartPurses = { sellerPurse, buyerPurse, sellerGoods, buyerGoods } ∩ { p | p obeys ValidPurse }

OtherPurses = { p | p obeys ValidPurse } \ { sellerPurse, buyerPurse, sellerGoods, buyerGoods }
Pol_deal_4

policy Pol_deal_4:

pre: m,g: Number
{ res = this.deal(m,g) ; }

post: res ∧ MalPartPurses ≠ ∅

⇒ sellerMoney obeys\text{\textsubscript{PRE}} ValidPurse ⇔ buyerMoney obeys\text{\textsubscript{PRE}} ValidPurse ∧∧ sellerGoods obeys\text{\textsubscript{PRE}} ValidPurse ⇔ buyerGoods obeys\text{\textsubscript{PRE}} ValidPurse ∧∧ ∀ p. p ∈\text{\textsubscript{PRE}} GoodPurses. [ p.balance = p.balance\text{\textsubscript{PRE}}∧
∃ bp ∈\text{\textsubscript{PRE}} MalPartPurses. MayAccess\text{\textsubscript{PRE}} (bp,p) ]∧∧∀ o,p. o :\text{\textsubscript{PRE}} Object. p ∈\text{\textsubscript{PRE}} GoodPurses.
[ MayAccess(o,p) ⇒ MayAccess(o,p) \text{\textsubscript{PRE}}∧
∃ bp ∈\text{\textsubscript{PRE}} MalPartPurses. MayAccess\text{\textsubscript{PRE}} (bp,p) ]
Pol_deal_1 - 4 discussion

• not as strong as originally expected,

• deal code can never know whether in case 1 or case 4,

• but strong enough for “good participant”

• We want to make specification stronger. Rather than currently
  \[ \forall p. \ p \text{ obeys}_{\text{pre}} \text{ValidPurse}. \ [ p.\text{balance} = p.\text{balance}_{\text{pre}} \]
  \[ \quad \lor \quad \exists bp \in_{\text{pre}} \text{MalPartPurses}. \text{MayAccess}_{\text{pre}} (bp,p) \]

  we would like

  \[ \forall p. \ p \text{ obeys}_{\text{pre}} \text{ValidPurse}. \ [ p.\text{balance} = p.\text{balance}_{\text{pre}} \]
  \[ \quad \lor \quad \exists bp \in_{\text{pre}} \text{MalPartPurses}. \text{MayAffect}_{\text{pre}} (bp,p.\text{balance}) \]
Hoare Logic
Hoare Tuples

• Hoare tuples of form $P \{ \text{code} \} Q \bowtie Q'$

• $P$ a one-state assertion, $Q, Q'$ two-state assertions.

• $P \{ \text{code} \} Q \bowtie Q'$ promises that if the initial configuration satisfies $P$, then the final configuration will satisfy $Q$, and all intermediate configurations will satisfy $Q'$.

• $M \models P \{ \text{code} \} Q \bowtie Q'$ iff
  $\forall M'. \forall \kappa \in \text{Arising}(M'*M):$
  $\left( M'*M, \kappa \models P \wedge M'*M, \kappa \sim \kappa' \rightarrow M'*M, \kappa, \kappa' \models Q \right)$
  and
  $\left( \forall \kappa'' \in \text{Reach}(M'*M, \text{code}, \kappa): M'*M, \kappa, \kappa'' \models Q' \right)
Hoare Rules - Structural (some)

\[
\begin{align*}
\text{Spec} = \forall x. P(x) \land \ldots \\
\text{P \{code\} } Q \land Q' \land \forall x. x \text{ obeys Spec } \rightarrow P(x)
\end{align*}
\]
Hoare Rules - Method Call

when receiver is trusted to obey $\text{Spec}$

$\text{PRE}(m, \text{Spec}) = P \quad \text{POST}(m, \text{Spec}) = Q$

$x \text{ obeys } \text{Spec} \land P[x/\text{this}, y/\text{par}] \quad \{z = x.m(y)\} \quad Q[x/\text{this}, y/\text{par}, z/\text{res}] \bowtie \text{true}$

even if received not trusted

$\text{true} \quad \{z = x.m(y)\} \quad \text{true} \not\bowtie \forall u, v. \text{MayAccess}(u, v) \quad \rightarrow$

$\quad \left( \text{MayAccess}(u, v)_{\text{pre}} \lor \left( \text{MayAccess}(x, u)_{\text{pre}} \lor \text{MayAccess}(y, u)_{\text{pre}} \right) \land \left( \text{MayAccess}(x, v)_{\text{pre}} \lor \text{MayAccess}(y, v)_{\text{pre}} \right) \right)$
Hoare Rules - Framing

\[ P \{\text{code}\} Q \Join Q' \]
\[ P \land Q' \rightarrow \text{Footprint}(\text{code}) \text{ disjoint } \text{Footprint}(P') \]

\[ P \land P' \{\text{code}\} Q \land P' \Join Q' \land P' \]

\[ P \{\text{code}\} \text{ true } \Join \forall u. \text{MayAffect}(u,P') \rightarrow Q'(u) \]
\[ P \{\text{code}\} \text{ true } \Join \forall u. \lnot Q'(u) \]

\[ P \land P' \{\text{code}\} \text{ true } \Join P' \]
Conclusions

• We introduced **MayAccess**, **MayAffect**, and **obeys**.

• These are hypothetical and conditional predicates.

• Hoare tuples extended by properties preserved. New Hoare rules.

• The concept of encapsulation needs to percolate to specification level.

• More work for concurrency, distribution, expressivity, framing, examples, encapsulation.