Towards Reasoning in the presence of code of unknown provenance

- or, trust and risk in an open world -

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

Hoare Rules - Method Call

In this talk, we argue:

- We can do better than that.
- It is important to do better than that.

And if I know nothing about receiver?

\[
\begin{align*}
\text{true } \{ \text{z = x.m(y)} \} & \text{ true}
\end{align*}
\]
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

  • **obeys** predicate: an object adheres to a specification,
  
  • **MayAccess** predicate: an object may read some property
  
  • **MayAffect** predicate: an object may affect some property

• Predicates **obeys**, **MayAccess** and **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).
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.

• 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).
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**

$M \models Policy$ iff

$\forall M'$. 

$\forall \kappa \in Arising(M'*M): M'*M, \kappa \models Policy$

$M'$ represents the “adversary”.

$Arising(M'*M)$ 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 ($\text{Policy}$).

Definition

$$M, \kappa \models o \text{ obeys } \text{Spec} \iff \forall \text{ Policy } \in \text{Spec.}$$

$$\forall \kappa' \in \text{Reach}(M, \kappa): M, \kappa' \models \text{Policy}[o/\text{this}]$$

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

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

**policy** `Pol_deal_1`:

**pre**: ....

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

**post**: ....

\( \forall p. \text{ p obeys ValidPurse } .... \)

\[ \begin{align*}
\text{ p.balance } &= \text{ p.balance}_{\text{PRE}} \lor \\
\exists bp. \ldots \neg (bp \text{ obeys ValidPurse}) \land \text{ MayAccess}_{\text{PRE}} (bp,p)
\end{align*} \]
Escrow Agent - Our running example
Escrow Agent - Remit
(proposed by Miller, van Cutsem, Tulloh, ESOP 2013)

• Buyer and Seller want to exchange g goods for m 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 $g$ goods for $m$ money

1. pay $m$ to escrowMoney from buyerMoney

2. if no success then exit

   // sufficient money

3. pay $g$ to escrowGoods from sellerGoods

4. if no success then
   pay $m$ to buyerMoney from escrowMoney
   exit

   // sufficient money and goods

5. pay $g$ to buyerGoods from escrowGoods

6. pay $m$ to sellerMoney from escrowMoney
The Escrow purses

• intermediate store of money, and 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 $\neg\text{res}$ then exit

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

4. if $\neg\text{res}$ 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 g goods for m 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

true { escrowMoney.deposit() } true

How much damage can it make?
Escrow Agent - Second Attempt
Escrow Agent - Second Attempt summary

• 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: { res=this.deposit(prs, amt) }  
    res=true    implies   trust, enough funds, and transfer of amt

    policy Pol_deposit_2: { res=this.deposit(prs, amt) }  
    res=false   implies   no trust or not enough funds, and no transfer

    policy Pol_sprout:  { res=this.sprout( ) }  
    res is a new Purse of same trustworthiness

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

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 - protect_balance

balance cannot be affected, unless you hold the purse itself

policy Pol_protect_balance:

∀ p, o.

( p obeys ValidPurse ∧ o : Object. → [ MayAffect(o, p.balance) → 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

\[
\text{escrowMoney} = \text{sellerMoney}.\text{sprout}()
\]

// ∀p. p obeys^\text{PRE} \text{ValidPurse} →
// [ p.balance^\text{PRE}=p\.balance \lor
// \text{MayAccess}^\text{PRE}(\text{sellerMoney},p) \land \neg(\text{sellerMoney obeys ValidPurse}) ]

res = \text{escrowMoney}. \text{deposit} (\text{buyerMoney},0)

//....

if !res then exit  // ....

res = \text{buyerMoney}. \text{deposit} (\text{escrowMoney},0)

// ∀p. p obeys^\text{PRE} \text{ValidPurse} →
// [ p.balance^\text{PRE}=p\.balance \lor
// \text{MayAccess}^\text{PRE}(\text{sellerMoney},p) \land \neg(\text{sellerMoney obeys ValidPurse}) \lor
// \text{MayAccess}^\text{PRE}(\text{buyerMoney},p) \land \neg(\text{buyerMoney obeys ValidPurse}) ]

if !res then exit  // ....

res = \text{escrowMoney}. \text{deposit} (\text{buyerMoney},0)

//....

if !res then exit

// ∀p. p obeys^\text{PRE} \text{ValidPurse} →
// [ p.balance^\text{PRE}=p\.balance \lor
// \text{MayAccess}^\text{PRE}(\text{sellerMoney},p) \land \neg(\text{sellerMoney obeys ValidPurse}) \lor
// \text{MayAccess}^\text{PRE}(\text{buyerMoney},p) \land \neg(\text{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
For \( \text{res} := \text{this}.\text{deal}(m,g) \) we have four cases:

**Policy Pol_deal_1:**
\[
\text{res} \land \text{buyer and seller “are good”} \implies
\]

“Surprises”:
- deal method not as “risk-free” as expected
- \( \text{res}=\text{true} \) does not imply successful transaction, nor that participants were good.

no Purse affected unless malicious participant had access before

**Policy Pol_deal_4:**
\[
\text{res} \land \text{buyer and seller “are bad”} \implies
\]

no Purse affected unless malicious participant had access before
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
  
  $\forall M'. \forall \kappa \in \text{Arising}(M'*M)$:
  
  ( $M'*M, \kappa \models P \land M'*M, \kappa \rightsquigarrow \kappa'$
  
  $\rightarrow M'*M, \kappa, \kappa' \models Q$ )

  and

  ( $\forall \kappa'' \in \text{Reach}(M'*M, \text{code}, \kappa): M'*M, \kappa, \kappa'' \models Q'$ )
Hoare Rules - Structural (some)

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

\[ P \{\text{code}\} Q \Join Q' \]
\[ \text{Spec} = \text{spec}\{ \text{Pol}_1, \ldots, \text{Pol}_i, \ldots, \text{Pol}_n \} \]
\[ P \{\text{code}\} Q \Join Q' \land \forall x. x \text{ obeys Spec} \rightarrow \text{Pol}_i[x/\text{this}] \]

\[ P \{\text{code}\} Q \Join Q' \]
\[ P' \rightarrow P \quad Q \rightarrow Q'' \quad Q' \rightarrow Q''' \]
\[ P' \{\text{code}\} Q'' \Join Q''' \]

\[ P \{\text{code}\} Q \Join Q'' \]
\[ P', Q' \rightarrow P \]
\[ P' \{\text{code}\} Q' \rightarrow Q \Join Q'''' \]

\[ P' \rightarrow P \quad \text{iff} \quad \kappa \models P' \rightarrow \kappa \models P \]
\[ Q \rightarrow Q'' \quad \text{iff} \quad \kappa, \kappa' \models Q \implies \kappa, \kappa' \models Q'' \]

\[ P', Q' \rightarrow P \quad \text{iff} \quad \kappa \models P' \land \kappa, \kappa' \models Q' \implies \kappa \models P \]
Hoare Rules - Method Call

when receiver is trusted to obey Spec

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

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

and regardless of whether receiver is trusted

\[ \text{true} \{ z = x.m(y) \} \text{ true} \not\bowtie \forall u, v. \ MayAccess(u, v) \rightarrow \]
\[ ( MayAccess(u, v) \text{ pre } \lor \]
\[ ( MayAccess(x, u) \text{ pre } \lor MayAccess(y, u) \text{ pre } ) \land \]
\[ ( MayAccess(x, v) \text{ pre } \lor MayAccess(y, v) \text{ pre } ) \]
Hoare Rules - Framing

\[
P \{ \text{code} \} Q \triangleleft Q' \\
\begin{align*}
P \land Q' & \rightarrow \text{Footprint(code)} \text{ disjoint Footprint}(P') \\
P \land P' \{ \text{code} \} Q \land P' \triangleleft Q' \land P'
\end{align*}
\]

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

\[
P \land P' \{ \text{code} \} \text{ true } \triangleleft P'
\]
Summary

• 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. More case studies. Ongoing design: refinement of the predicates, and new predicates.
Conclusions

In this talk, we argued:
- We can reason in the presence of “untrusted”/“unknown” code
- It is important to do that.

We need to specify

- what *will* happen,
- as well as what *will not* happen