Robust Safety for Move

Marco Patrignani\textsuperscript{1}  \hspace{1cm}  Sam Blackshear\textsuperscript{2}

\textsuperscript{1}Università di Trento
\textsuperscript{2}Mysten Labs

Interested? We're hiring!
Robust Safety for Move

Interested? We’re hiring!

1. Marco Patrignani
2. Sam Blackshear

Mysten Labs
The Move Language
Smart contract safety is an existential threat to broader crypto adoption

- 100M+ hacks are routine
- No reason to expect that future smart contract developer will do better...
- Safer SC languages, advanced testing/analysis/verification tools are the only way to grow the dev community in a sustainable way
Smart contracts are unconventional programs

- Smart contracts really only do three things:
  - Define new asset types
  - Read, write, and transfer assets
  - Check access control policies

Thus, need language support for
- Safe abstractions for custom assets, ownership, access control
- Strong isolation—writing safe open-source code that interacts **directly** with code written by motivated attackers

Not common tasks in conventional languages
Not well-supported by existing SC languages
In other smart contract langs, you typically cannot:

- Pass asset as an argument to a function, or return one from a function
- Store an asset in a data structure
- Let a callee function temporarily borrow an asset
- Declare an asset type in contract 1 that is used by contract 2
- Take an asset outside of the contract that created it
  - “trapped” forever in a hash table inside its defining contract

Assets, ownership are the fundamental building blocks of smart contracts, but there’s no vocabulary for describing them!

Move is the first smart contract language to tackle this problem.
Assets and ownership encoded via substructural types

“If you give me a coin, I will give you a car title”

fun buy(c: Coin): CarTitle

“If you show me your title and pay a fee, I will give you a car registration”

fun register(c: &CarTitle, fee: Coin): CarRegistration { ... }

CarTitle, CarRegistration, Coin are user-defined types declared in different modules. Can flow across trust boundaries without losing integrity
Type system prevents misuse of asset values

Protection against:

**Duplication**

```racket
fun f(c: Coin) {
    let x = copy c; // error
    let y = &c;
    let copied = *y; // error
}
```

**“Double-spending”**

```racket
fun h(c: Coin) {
    pay(move c);
    pay(move c); // error
}
```

**Destruction**

```racket
fun g(c: Coin) {
    c = ... ; // error
    return // error—must move c!
}
```

Ensures that digital assets behave like physical ones
Move design optimizes for safety + predictability

- No dynamic dispatch (no re-entrancy)
- No mixing of aliasing and mutability (like Rust)
- Type/memory/resource safety enforced by bytecode verifier
- Strong isolation aka “robust safety” by default
  - See upcoming CSF ‘23 paper
- Mathematically ill-defined ops (e.g., int overflow) abort: “SafeMath by default”
- Co-developed with the Move Prover formal verification tool (see CAV’20, TACAS ‘21 papers)
Contributions of this Work
Contributions

- formalise Robust Safety (RS) for Move
  - identify the prerequisites for RS
Contributions

• formalise Robust Safety (RS) for Move
  • identify the prerequisites for RS

• prove all Move programs attain RS
Contributions

• formalise **Robust Safety** (RS) for **Move**
  • identify the prerequisites for RS

• prove all **Move** programs attain RS

• implement and evaluate missing tool(s) for RS prerequisites
Contributions

• formalise Robust Safety (RS) for Move
  • identify the prerequisites for RS

• prove all Move programs attain RS

• implement and evaluate missing tool(s) for RS prerequisites
Contributions

Next:
- formalise Robust Safety (RS) for Move
- identify the prerequisites for RS

Then:
- prove all Move programs attain RS
- implement and evaluate missing tool(s) for RS prerequisites
Contributions

- formalise Robust Safety (RS) for Move
- identify the prerequisites for RS
- prove all Move programs attain RS
- implement and evaluate missing tool(s) for RS prerequisites
Robust Safety (for Move)
Robust Safety: maintaining key safety properties even when interacting with arbitrary untrusted code

Bengtson et al. TOPLAS’11, Gordon & Jeffrey JCS’03, Swasey et al. OOPSLA’17 and many more
What is Robust Safety?

Robust Safety: maintaining *key safety properties* even when interacting with *arbitrary untrusted code*

*Bengtson et al. TOPLAS’11, Gordon& Jeffrey JCS’03, Swasey et al. OOPSLA’17 and many more*

• key safety properties: programmer-inserted invariants
What is Robust Safety?

Robust Safety: maintaining key safety properties even when interacting with arbitrary untrusted code

Bengtson et al. TOPLAS’11, Gordon & Jeffrey JCS’03, Swasey et al. OOPSLA’17 and many more

- key safety properties: programmer-inserted invariants
- arbitrary untrusted code: active attacker (with code-like capabilities)
A (massaged!) Move Example

```rust
module NextCoin {

    struct Coin has key { value: u64 }
    struct Info has key { tot_supply: u64 }

    spec { ∀c: Coin, info.tot_supply = sum(c.value) }

    public fun mint(... , value: u64): Coin {
        let info = borrow_global_mut< Info> (...);
        info.tot_supply = info.tot_supply + value;
        Coin { value } // invariant broken and restored
    }

    public fun value_mut(coin: & mut Coin): & mut u64 {
        & mut coin.value // not robustly safe!
    }
}
```
Threat Model

- **trusted code**: the code with invariants (NextCoin)
Threat Model

- **trusted code**: the code with invariants (NextCoin)
- **attackers**: active, write code (e.g., other smart contracts) and interact with the trusted code to break safety
Threat Model

- **trusted code**: the code with invariants (NextCoin)
- **attackers**: active, write code (e.g., other smart contracts) and interact with the trusted code to break safety
- **safety**: specified by the programmer-inserted invariants (spec)
Local Invariant Verification

• spec holds for module NextCoin locally
Local Invariant Verification

• spec holds for module NextCoin locally
  verification done by
  • Move bytecode verifier
  • Move Prover

Blackshear et al. Whitepaper’19
Zhong et al. CAV’20
Local Invariant Verification

• spec holds for module NextCoin **locally**
• verification done by
  • Move bytecode verifier
  • Move Prover
• (when attackers are not considered)

```
spec { \forall c: Coin, info.tot_supply = \text{sum}(c.value) }

public fun mint(... , value: u64): Coin {
  let info = borrow_global_mut< Info> (...);
  info.tot_supply = info.tot_supply + value;
  Coin { value } // invariant broken and restored
}
```
Global Invariant Verification

```
spec { \forall c: Coin, info.tot_supply = sum(c.value) }

public fun value_mut(coin: &mut Coin): &mut u64 {
    &mut coin.value // not robustly safe!
}
```

- spec does not hold **globally** (when attackers are considered)
Global Invariant Verification

```
spec { \( \forall c: \text{Coin}, \text{info.tot\_supply} = \text{sum}(c.\text{value}) \) }

public fun value\_mut(coin: &mut Coin): &mut u64 {
    &mut coin.value // not robustly safe!
}
```

- spec does not hold **globally** (when attackers **are** considered)

```
fun attacker(c: &mut Coin) {
    let value\_ref = Coin::value\_mut(c);
    *value\_ref = *value\_ref + 1000; // violates spec!
}
```
• **Problem:** `value_mut` leaks an invariant-based value
• **Problem**: `value_mut` leaks an invariant-based value

• **Solution**: enforce *encapsulation* on invariant-based values
Problem: value_mut leaks an invariant-based value

Solution: enforce encapsulation on invariant-based values

Trivial? perhaps

Not-so-trivial? formalising the sufficient conditions for RS and designing an efficient analysis that checks these conditions
Robust Safety Definition

A Move module $\Omega$ with invariants $\iota$ has RS iff:

- $\Omega$ is well-typed

$\Lambda \vdash \text{loc } \Omega : \iota$

$\Xi \vdash \text{enc } \Omega : \iota$

$\forall A. \Omega \vdash A : \text{atk}$ running $\Omega$ and $A(\Omega + A) \leadsto$ respects $\iota$

What are $\Lambda$ and $\Xi$?
Robust Safety Definition

A Move module $\Omega$ with invariants $\iota$ has RS iff:

- $\Omega$ is well-typed
- $\Omega$ has verified $\iota$ locally
Robust Safety Definition

A Move module $\Omega$ with invariants $\iota$ has RS iff:

- $\Omega$ is well-typed
- $\Omega$ has verified $\iota$ locally
- $\Omega$ has encapsulated $\iota$

What are $\Lambda$ and $\Xi$?
A Move module $\Omega$ with invariants $\iota$ has RS iff:

- $\Omega$ is well-typed
- $\Omega$ has verified $\iota$ locally
- $\Omega$ has encapsulated $\iota$
- for all attackers $A$
  running $\Omega$ and $A$
  respects $\iota$

What are $\Lambda$ and $\Xi$?
Robust Safety Definition

A Move module $\Omega$ with invariants $\iota$ has RS iff:

- $\Omega$ is well-typed $\vdash \Omega : wt$
- $\Omega$ has verified $\iota$ locally $\Lambda \vdash \text{loc } \Omega : \iota$
- $\Omega$ has encapsulated $\iota$ $\Xi \vdash \text{enc } \Omega : \iota$
- for all attackers $A$
  - running $\Omega$ and $A$
  - respects $\iota$

What are $\Lambda$ and $\Xi$?
A Move module $\Omega$ with invariants $\iota$ has RS iff:

- $\Omega$ is well-typed: $\vdash \Omega : wt$
- $\Omega$ has verified $\iota$ locally: $\Lambda \vdash_{loc} \Omega : \iota$
- $\Omega$ has encapsulated $\iota$
- for all attackers $A$ running $\Omega$ and $A$ respects $\iota$
Robust Safety Definition

A Move module $\Omega$ with invariants $\iota$ has RS iff:

- $\Omega$ is well-typed 
  \[ \vdash \Omega : wt \]
- $\Omega$ has verified $\iota$ locally 
  \[ \Lambda \vdash_{loc} \Omega : \iota \]
- $\Omega$ has encapsulated $\iota$ 
  \[ \Xi \vdash_{enc} \Omega : \iota \]
- for all attackers $A$
  running $\Omega$ and $A$
  respects $\iota$
Robust Safety Definition

A Move module $\Omega$ with invariants $\iota$ has RS iff:

- $\Omega$ is well-typed $\vdash \Omega : wt$
- $\Omega$ has verified $\iota$ locally $\Lambda \vdash_{loc} \Omega : \iota$
- $\Omega$ has encapsulated $\iota$ $\Xi \vdash_{enc} \Omega : \iota$
- for all attackers $A$ running $\Omega$ and $A$ respects $\iota$ $\forall A. \Omega \vdash A : atk$
Robust Safety Definition

A Move module $\Omega$ with invariants $\iota$ has RS iff:

- $\Omega$ is well-typed $\vdash \Omega : wt$
- $\Omega$ has verified $\iota$ locally $\Lambda \vdash_{loc} \Omega : \iota$
- $\Omega$ has encapsulated $\iota$ $\Xi \vdash_{enc} \Omega : \iota$
- for all attackers $A$
  - running $\Omega$ and $A$
  - respects $\iota$
  $\forall A. \Omega \vdash A : atk$
  $(\Omega + A) \leadsto \overline{\alpha}$
A Move module $\Omega$ with invariants $\iota$ has RS iff:

- $\Omega$ is well-typed
- $\Omega$ has verified $\iota$ locally
- $\Omega$ has encapsulated $\iota$
- for all attackers $A$

running $\Omega$ and $A$
respects $\iota$

\[ \vdash \Omega : wt \]
\[ \Lambda \vdash_{loc} \Omega : \iota \]
\[ \Xi \vdash_{enc} \Omega : \iota \]
\[ \forall A. \Omega \vdash A : atk \]
\[ (\Omega + A) \leadsto \overline{\alpha} \]
\[ \overline{\alpha} \not\vdash \iota \]
A Move module $\Omega$ with invariants $\iota$ has RS iff:

- $\Omega$ is well-typed $\vdash_{\omega} \Omega : wt$
- $\Omega$ has verified $\iota$ locally $\Lambda \vdash_{loc} \Omega : \iota$
- $\Omega$ has encapsulated $\iota$ $\Xi \vdash_{enc} \Omega : \iota$
- for all attackers $A$ running $\Omega$ and $A$ respects $\iota$ $\forall A. \Omega \vdash A : \text{atk}$
  $$(\Omega + A) \rightsquigarrow \overline{\alpha}$$
  $\overline{\alpha} \vdash \iota$

what are $\Lambda$ and $\Xi$?
Tools for Robust Safety in Move
Only who declares `Coin` can:

- Create a value of type `Coin`
- “Unpack” a `Coin` into its field(s)
- Acquire a reference to a field of `Coin` via a Rust-style mutable or immutable borrow
• assume invariants specified by the programmer hold at the entry of each public function
• ensure that they continue to hold at the exit
Move Prover for Local Invariants

• assume invariants specified by the programmer hold at the entry of each public function

• ensure that they continue to hold at the exit

```rust
spec { ∀c: Coin, info.tot_supply = sum(c.value) }

public fun mint(... , value: u64): Coin {
  Coin { value } // invariant broken
}
```
• Two classes of attackers:
  • Blockchain-based (imm)
  • non Blockchain-based (mut)
Encapsulator(s) for ... Encapsulation

- Two classes of attackers:
  - Blockchain-based (imm)
  - non Blockchain-based (mut)

- encapsulation:
  when control goes to the attacker
Encapsulator(s) for ... Encapsulation

• Two classes of attackers:
  • Blockchain-based (imm)
  • non Blockchain-based (mut)

• encapsulation:
  when control goes to the attacker

any resource with an invariant
• Two classes of attackers:
  • Blockchain-based (imm)
  • non Blockchain-based (mut)

• encapsulation:
  when control goes to the attacker

any resource with an invariant

is not accessible to the attacker
Encapsulator(s) for ... Encapsulation

- Two classes of attackers:
  - Blockchain-based (imm)
  - non Blockchain-based (mut)
-封装: when control goes to the attacker
calls (mut) and returns (imm & mut)
any resource with an invariant
is not accessible to the attacker
Encapsulator(s) for ... Encapsulation

- Two classes of attackers:
  - Blockchain-based (imm)
  - non Blockchain-based (mut)
- **encapsulation:**
  when control goes to the attacker
  calls (mut) and returns (imm & mut)
  any resource with an invariant
  using abstract values \( \hat{v} \)
  is not accessible to the attacker
Two classes of attackers:
- Blockchain-based (imm)
- non Blockchain-based (mut)

**encapsulation:**
when control goes to the attacker calls (mut) and returns (imm & mut)
any resource with an invariant using abstract values \( \hat{v} \)
is not accessible to the attacker
any relevant \( \hat{v} \) is not in \( A \)'s state
Encapsulator Details

• static intraprocedural escape analysis
• abstract values $\hat{v} \in \{\text{NonRef, OkRef, InvRef}\}$
  • $\text{NonRef} \subseteq \text{InvRef} \quad \text{OkRef} \subseteq \text{InvRef}$
Encapsulator Details

- static intraprocedural escape analysis
- abstract values $\hat{v} \in \{\text{NonRef}, \text{OkRef}, \text{InvRef}\}$
  - $\text{NonRef} \sqsubseteq \text{InvRef}$
  - $\text{OkRef} \sqsubseteq \text{InvRef}$

\[
\left(\Xi_{\text{imm}}\text{-BorrowFld-Relevant}\right)
\]

\[
\begin{align*}
\Omega, P, \iota, \text{BorrowFld} & \vdash \langle \hat{L}, \hat{v}::\hat{S} \rangle \leadsto \langle \hat{L}, \text{InvRef}::\hat{S} \rangle \\
\end{align*}
\]
Encapsulator Details

- static intraprocedural escape analysis
- abstract values $\hat{v} \in \{\text{NonRef}, \text{OkRef}, \text{InvRef}\}$
  - $\text{NonRef} \sqsubseteq \text{InvRef}$
  - $\text{OkRef} \sqsubseteq \text{InvRef}$

\[
\begin{align*}
(\Xi_{\text{imm}}\text{-BorrowFld- Relevant}) \\
\quad f \in \iota
\end{align*}
\]
\[
\Omega, P, \iota, \text{BorrowFld} \langle f \rangle \vdash \langle \hat{L}, \hat{v}::\hat{S} \rangle \leadsto \langle \hat{L}, \text{InvRef}::\hat{S} \rangle
\]
\[
(\Xi_{\text{imm}}\text{-BorrowFld-Irrelevant}) \\
\quad f \notin \iota
\]
\[
\Omega, P, \iota, \text{BorrowFld} \langle f \rangle \vdash \langle \hat{L}, \hat{v}::\hat{S} \rangle \leadsto \langle \hat{L}, \hat{v}::\hat{S} \rangle
\]
Encapsulator Details

- static intraprocedural escape analysis
- abstract values $\hat{v} \in \{\text{NonRef}, \text{OkRef}, \text{InvRef}\}$
  - $\text{NonRef} \sqsubseteq \text{InvRef} \quad \text{OkRef} \sqsubseteq \text{InvRef}$

\[
\begin{align*}
\text{(Imm-BorrowFld-Relevant)} & \quad f \in \iota \\
\Omega, P, \iota, \text{BorrowFld} \vdash \langle \hat{L}, \hat{v} :: \hat{S} \rangle \leadsto \langle \hat{L}, \text{InvRef} :: \hat{S} \rangle \\
\text{(Imm-BorrowFld-Irrelevant)} & \quad f \notin \iota \\
\Omega, P, \iota, \text{BorrowFld} \vdash \langle \hat{L}, \hat{v} :: \hat{S} \rangle \leadsto \langle \hat{L}, \hat{v} :: \hat{S} \rangle \\
\text{(Imm-Return)} & \quad \|\Omega(P)\text{.retry}\| = n \quad \forall i \in 1..n. \hat{v}_i \neq \text{InvRef} \\
\Omega, P, \iota, \text{Ret} \vdash \langle \hat{L}, \hat{v}_1 :: \hat{v}_n :: \hat{S} \rangle \leadsto \langle \hat{L}, \hat{v}_1 :: \hat{v}_n :: \hat{S} \rangle
\end{align*}
\]
### Encapsulator Evaluation

<table>
<thead>
<tr>
<th>Bench</th>
<th>Mod</th>
<th>Fun</th>
<th>Rec</th>
<th>Instr</th>
<th>Err</th>
<th>$T_p$</th>
<th>$T_e$</th>
</tr>
</thead>
<tbody>
<tr>
<td>starcoin</td>
<td>60</td>
<td>431</td>
<td>88</td>
<td>8243</td>
<td>2</td>
<td>3178</td>
<td>10</td>
</tr>
<tr>
<td>diem</td>
<td>13</td>
<td>102</td>
<td>19</td>
<td>1830</td>
<td>0</td>
<td>1651</td>
<td>1</td>
</tr>
<tr>
<td>mai</td>
<td>45</td>
<td>411</td>
<td>77</td>
<td>7881</td>
<td>0</td>
<td>4209</td>
<td>12</td>
</tr>
<tr>
<td>bridge</td>
<td>36</td>
<td>352</td>
<td>85</td>
<td>8060</td>
<td>0</td>
<td>2428</td>
<td>8</td>
</tr>
<tr>
<td>blackhole</td>
<td>36</td>
<td>324</td>
<td>72</td>
<td>6030</td>
<td>0</td>
<td>2289</td>
<td>7</td>
</tr>
<tr>
<td>alma</td>
<td>35</td>
<td>333</td>
<td>67</td>
<td>6318</td>
<td>0</td>
<td>2102</td>
<td>8</td>
</tr>
<tr>
<td>starswap</td>
<td>33</td>
<td>335</td>
<td>67</td>
<td>6617</td>
<td>0</td>
<td>14993</td>
<td>7</td>
</tr>
<tr>
<td>meteor</td>
<td>32</td>
<td>323</td>
<td>69</td>
<td>5981</td>
<td>0</td>
<td>1641</td>
<td>7</td>
</tr>
<tr>
<td>taohe</td>
<td>11</td>
<td>40</td>
<td>7</td>
<td>305</td>
<td>0</td>
<td>1022</td>
<td>1</td>
</tr>
<tr>
<td>stdlib</td>
<td>9</td>
<td>66</td>
<td>5</td>
<td>933</td>
<td>1</td>
<td>1151</td>
<td>1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>310</strong></td>
<td><strong>2717</strong></td>
<td><strong>556</strong></td>
<td><strong>52198</strong></td>
<td><strong>3</strong></td>
<td><strong>34664</strong></td>
<td><strong>62</strong></td>
</tr>
</tbody>
</table>
## Encapsulator Evaluation

<table>
<thead>
<tr>
<th>Bench</th>
<th>Mod</th>
<th>Fun</th>
<th>Rec</th>
<th>Instr</th>
<th>Err</th>
<th>$T_p$</th>
<th>$T_e$</th>
</tr>
</thead>
<tbody>
<tr>
<td>starcoin</td>
<td>60</td>
<td>431</td>
<td>88</td>
<td>8243</td>
<td>2</td>
<td>3178</td>
<td>10</td>
</tr>
<tr>
<td>diem</td>
<td>13</td>
<td>102</td>
<td>19</td>
<td>1830</td>
<td>0</td>
<td>1651</td>
<td>1</td>
</tr>
<tr>
<td>mai</td>
<td>45</td>
<td>411</td>
<td>77</td>
<td>7881</td>
<td>0</td>
<td>4209</td>
<td>12</td>
</tr>
<tr>
<td>bridge</td>
<td>36</td>
<td>352</td>
<td>85</td>
<td>8060</td>
<td>0</td>
<td>2428</td>
<td>8</td>
</tr>
<tr>
<td>blackhole</td>
<td>36</td>
<td>324</td>
<td>72</td>
<td>6030</td>
<td>0</td>
<td>2289</td>
<td>7</td>
</tr>
<tr>
<td>alma</td>
<td>35</td>
<td>333</td>
<td>67</td>
<td>6318</td>
<td>0</td>
<td>2102</td>
<td>8</td>
</tr>
<tr>
<td>starswap</td>
<td>33</td>
<td>335</td>
<td>67</td>
<td>6617</td>
<td>0</td>
<td>14993</td>
<td>7</td>
</tr>
<tr>
<td>meteor</td>
<td>32</td>
<td>323</td>
<td>69</td>
<td>5981</td>
<td>0</td>
<td>1641</td>
<td>7</td>
</tr>
<tr>
<td>taohe</td>
<td>11</td>
<td>40</td>
<td>7</td>
<td>305</td>
<td>0</td>
<td>1022</td>
<td>1</td>
</tr>
<tr>
<td>stdlib</td>
<td>9</td>
<td>66</td>
<td>5</td>
<td>933</td>
<td>1</td>
<td>1151</td>
<td>1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>310</td>
<td>2717</td>
<td>556</td>
<td>52198</td>
<td>3</td>
<td>34664</td>
<td>62</td>
</tr>
</tbody>
</table>
Questions?