Towards Security Analysis of Ethereum smart contracts
Motivation

Blockchain

Blockchain-based Venture Capital Fund Hacked for $60 Million

David Z. Morris
Jun 18, 2016

News emerged Friday that The DAO, a venture capital fund operating through a decentralized blockchain inspired by Bitcoin, had been robbed of more than $60 million worth of Ether digital currency, or about 1/3 of its value, through a code exploit. The DAO, which raised more than $150 million in May, had been intended as a showcase for the potential of Ethereum, a blockchain platform for cloud-based financial agreements.

The nature of the hack was outlined in an open letter claiming to be from the attacker, posted to Pastebin this morning. In part, it reads:
Ethereum - High level

- transfer money to A
- call the contract of B
- create a contract X
Overview on Ethereum

External Account
0xa30...

Balance: 5 Eth
# Overview on Ethereum

<table>
<thead>
<tr>
<th>External Account</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0d0...</td>
<td></td>
</tr>
<tr>
<td>Balance: 1 Eth</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>External Account</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0xa30...</td>
<td></td>
</tr>
<tr>
<td>Balance: 5 Eth</td>
<td></td>
</tr>
</tbody>
</table>
Overview on Ethereum

External Account
0xdc0...
Balance: 2 Eth

Transfer
1Eth

External Account
0xa30...
Balance: 4 Eth
Overview on Ethereum

External Account
0xdc0…
Balance: 2 Eth

Transfer
1Eth

Contract Account
0xde2…
Balance: 1 Eth

External Account
0xa30…
Balance: 4 Eth
Overview on Ethereum

External Account

0xdc0...
Balance: 2 Eth

Contract Account

0xde2...
Balance: 1 Eth

Create new contract

0xfe0...
Balance: 3 Eth

Transfer

1Eth

External Account

0xa30...
Balance: 1 Eth

Balance: 1 Eth

Balance: 3 Eth

3Eth
Overview on Ethereum

External Account
0xdec0...
Balance: 2 Eth

Contract Account
0xde2...
Balance: 2 Eth

Contract Account
0xfe0...
Balance: 3 Eth

Transfer
1 Eth

Call contract
1 Eth

Create new contract
3 Eth
Overview on Ethereum

External Account
0xdec0...
Balance: 2 Eth

Contract Account
0xde2...
Balance: 2 Eth

Transfer
1Eth

Call Contract
1Eth

Create new contract
3Eth

External Account
0xda30...
Balance: 0 Eth

Contract Account
0xfe0...
Balance: 3 Eth
EVM bytecode

- Quasi Turing-Complete language
  - computation bounded by 'gas'
- General architecture: Stack machine
- Standard operations for stack/memory/storage manipulation
- Operations to initiate new internal transactions
  - calling contracts (CALL, CALLCODE, ...)
  - creating new contracts (CREATE)
- Operations for accessing the environment
  - information on the current call (CALLER, ...)
  - information on the state of the accounts
Example - CALL

PUSH1 0x01
PUSH1 0x60
MSTORE
PUSH1 0x20
PUSH1 0x40
PUSH1 0x01
PUSH1 0x60
PUSH1 0x00
PUSH32 0x0318247CB34f134f3cF49E97647227dc2D75Abe8
GAS
CALL
Example - CALL

PUSH1 0x01
PUSH1 0x60
MSTORE
PUSH1 0x20
PUSH1 0x40
PUSH1 0x01
PUSH1 0x60
PUSH1 0x00
PUSH32 0x0318247CB34f134f3cF49E97647227dc2D75Abe8
GAS
CALL
Example - CALL

PUSH1 0x01
PUSH1 0x60
MSTORE
PUSH1 0x20
PUSH1 0x40
PUSH1 0x01
PUSH1 0x60
PUSH1 0x00
PUSH32 0x0318247CB34f134f3cF49E97647227dc2D75Abe8
GAS
CALL
Example - CALL

```
PUSH1 0x01
PUSH1 0x60
MSTORE
PUSH1 0x20
PUSH1 0x40
PUSH1 0x01
PUSH1 0x60
PUSH1 0x00
PUSH32 0x0318247CB34f134f3cF49E97647227dc2D75Abe8
GAS
CALL
```
Example - CALL

PUSH1 0x01
PUSH1 0x60
MSTORE
PUSH1 0x20
PUSH1 0x40
PUSH1 0x01
PUSH1 0x60
PUSH1 0x00
PUSH32 0x0318247CB34f134f3cF49E97647227dc2D75Abe8
GAS
CALL
Example - CALL

PUSH1 0x01
PUSH1 0x60
MSTORE
PUSH1 0x20
PUSH1 0x40
PUSH1 0x01
PUSH1 0x60
PUSH1 0x00
PUSH32 0x0318247CB34f134f3cF49E97647227dc2D75Abe8
GAS
CALL
Example - CALL

PUSH1 0x01
PUSH1 0x60
MSTORE
PUSH1 0x20
PUSH1 0x40
PUSH1 0x01
PUSH1 0x60
PUSH1 0x00
PUSH32 0x0318247CB34f134f3CF49E97647227dc2D75Abe8
GAS
CALL
Example - CALL

PUSH1 0x01
PUSH1 0x60
MSTORE
PUSH1 0x20
PUSH1 0x40
PUSH1 0x01
PUSH1 0x60
PUSH1 0x00
PUSH32 0x0318247CB34f134f3cF49E97647227dc2D75Abe8
GAS
CALL
Example - CALL

PUSH1 0x01
PUSH1 0x60
MSTORE
PUSH1 0x20
PUSH1 0x40
PUSH1 0x01
PUSH1 0x60
PUSH1 0x00
PUSH32 0x0318247CB34f134f3cF49E97647227dc2D75Abe8
GAS
CALL
Example - CALL

PUSH1 0x01
PUSH1 0x60
MSTORE
PUSH1 0x20
PUSH1 0x40
PUSH1 0x01
PUSH1 0x60
PUSH1 0x00
PUSH32 0x0318247CB34f134f3cF49E97647227dc2
GAS
CALL

CALL
Example - CALL

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUSH1 0x01</td>
<td>0x01</td>
</tr>
<tr>
<td>PUSH1 0x60</td>
<td>0x60</td>
</tr>
<tr>
<td>MSTORE</td>
<td>0x0318247CB34f134f3cF49E97647227dc2</td>
</tr>
<tr>
<td>PUSH1 0x20</td>
<td>0</td>
</tr>
<tr>
<td>PUSH1 0x40</td>
<td>96</td>
</tr>
<tr>
<td>PUSH1 0x01</td>
<td>1</td>
</tr>
<tr>
<td>PUSH1 0x60</td>
<td>64</td>
</tr>
<tr>
<td>PUSH1 0x00</td>
<td>32</td>
</tr>
<tr>
<td>PUSH32 0x0318247CB34f134f3cF49E97647227dc2D75Abe8</td>
<td>1000</td>
</tr>
</tbody>
</table>

GAS
CALL
Example - CALL

PUSH1 0x01
PUSH1 0x60
MSTORE
PUSH1 0x20
PUSH1 0x40
PUSH1 0x01
PUSH1 0x60
PUSH1 0x00
PUSH32 0x0318247CB34f134f3cF49E97647227dc2D75Abe8
GAS
CALL

Example:

- **PUSH1 0x01**: Push 1 byte with value 0x01
- **PUSH1 0x60**: Push 1 byte with value 0x60
- **MSTORE**: Store the value
- **PUSH1 0x20**: Push 1 byte with value 0x20
- **PUSH1 0x40**: Push 1 byte with value 0x40
- **PUSH1 0x01**: Push 1 byte with value 0x01
- **PUSH1 0x60**: Push 1 byte with value 0x60
- **PUSH1 0x00**: Push 1 byte with value 0x00
- **PUSH32 0x0318247CB34f134f3cF49E97647227dc2D75Abe8**: Push 32 bytes with value 0x0318247CB34f134f3cF49E97647227dc2D75Abe8
- **GAS**: Gas consumption
- **CALL**: Call function

**Return Value Details**:
- **Gas to call**
- **Recipient address**
- **Amount of wei to transfer**
- **Input value address**
- **Input value size**
- **Return value address**
- **Return value size**
The DAO

```solidity
contract DAO {
    mapping (address => uint) donations;

    function donate() {
        donations[msg.sender] += msg.value;
    }

    function withdraw() {
        if (donations[msg.sender] > 0) {
            msg.sender.call.value(donations[msg.sender])();
            donations[msg.sender] = 0;
        }
    }
}
```

- mapping keeping track of the donations made by different addresses
- function for performing donations
- function for withdrawing donations
The DAO

contract DAO {
    mapping (address => uint) donations;

    function donate() {
        donations[msg.sender] += msg.value;
    }

    function withdraw() {
        if (donations[msg.sender] > 0) {
            msg.sender.call.value(donations[msg.sender]());
            donations[msg.sender] = 0;
        }
    }
}

mapping keeping track of the donations made by different addresses

function for performing donations

function for withdrawing donations

DAO contract

0xde2...

::

    guard?

    CALL

    invalidate guard

::
The DAO

```solidity
contract DAO {
    mapping (address => uint) donations;

    function donate() {
        donations[msg.sender] += msg.value;
    }

    function withdraw() {
        if (donations[msg.sender] > 0) {
            msg.sender.call.value(donations[msg.sender])();
            donations[msg.sender] = 0;
        }
    }
}
```

- `mapping (address => uint) donations;` - mapping keeping track of the donations made by different addresses
- `function donate() { donations[msg.sender] += msg.value; }` - function for performing donations
- `function withdraw() { if (donations[msg.sender] > 0) { msg.sender.call.value(donations[msg.sender])(); donations[msg.sender] = 0; } }` - function for withdrawing donations
contract DAO {
    mapping (address => uint) donations;

    function donate() {
        donations[msg.sender] += msg.value;
    }

    function withdraw() {
        if (donations[msg.sender] > 0) {
            msg.sender.call.value(donations[msg.sender])();
            donations[msg.sender] = 0;
        }
    }
}
The DAO

```solidity
contract DAO {
    mapping (address => uint) donations;

    function donate() {
        donations[msg.sender] += msg.value;
    }

    function withdraw()
    { if (donations[msg.sender] > 0) {
        msg.sender.call.value(donations[msg.sender])();
        donations[msg.sender] = 0;
    }
}
```

- `contract DAO`: Defines the DAO contract.
- `mapping (address => uint) donations;`: Tracks donations made by different addresses.
- `function donate()`: Performs donations.
- `function withdraw()`: Withdraws donations.

---

DAO contract

- `0xde2...`: Address of the DAO contract.
- `guard?`: Guard condition for the contract.
- `CALL`: Call operation within the contract.
- `invalidate guard`: Invalidate the guard condition.
Contract interactions

```solidity
contract DAO {
    mapping (address => uint) donations;

    function donate() {
        donations[msg.sender] += msg.value;
    }

    function withdraw() {
        if (donations[msg.sender] > 0) {
            msg.sender.call.value(donations[msg.sender])();
            donations[msg.sender] = 0;
        }
    }
}
```

DAO contract

0xde2...

![Diagram of a DAO contract with a piggy bank, a database, an arrow labeled "guard?", and another arrow labeled "inactivate guard".]
contract DAO {
    mapping (address => uint) donations;

    function donate() {
        donations[msg.sender] += msg.value;
    }

    function withdraw() {
        if (donations[msg.sender] > 0)
            msg.sender.call.value(donations[msg.sender])();
        donations[msg.sender] = 0;
    }
}
contract DAO {
    mapping (address => uint) donations;

    function donate() {
        donations[msg.sender] += msg.value;
    }

    function withdraw() {
        if (donations[msg.sender] > 0) {
            msg.sender.call.value(donations[msg.sender])();
            donations[msg.sender] = 0;
        }
    }
}
Contract interactions

```solidity
contract DAO {
    mapping (address => uint) donations;

    function donate() {
        donations[msg.sender] += msg.value;
    }

    function withdraw() {
        if (donations[msg.sender] > 0) {
            msg.sender.call.value(donations[msg.sender])();
            donations[msg.sender] = 0;
        }
    }
}
```

Attacker contract

1 Eth (donate)

---

DAO contract

guard: true

guard?

CALL

invalidate guard

CALL

---

0xde2...

0xfc2...
contract DAO {
    mapping (address => uint) donations;

    function donate() {
        donations[msg.sender] += msg.value;
    }

    function withdraw(){
        if (donations[msg.sender] > 0)
        {
            msg.sender.call.value(donations[msg.sender])();
            donations[msg.sender] = 0;
        }
    }
}
```
contract DAO {
    mapping (address => uint) donations;

    function donate() {
        donations[msg.sender] += msg.value;
    }

    function withdraw() {
        if (donations[msg.sender] > 0) {
            msg.sender.call.value(donations[msg.sender]());
            donations[msg.sender] = 0;
        }
    }
}
```
contract DAO {
    mapping (address => uint) donations;

    function donate() {
        donations[msg.sender] += msg.value;
    }

    function withdraw() {
        if (donations[msg.sender] > 0) {
            msg.sender.call.value(donations[msg.sender])();
            donations[msg.sender] = 0;
        }
    }
}
Contract interactions

```solidity
contract DAO {
    mapping (address => uint) donations;

    function donate() {
        donations[msg.sender] += msg.value;
    }

    function withdraw() {
        if (donations[msg.sender] > 0)
            msg.sender.call.value(donations[msg.sender])();
        donations[msg.sender] = 0;
    }
}
```
contract DAO {
    mapping (address => uint) donations;

    function donate() {
        donations[msg.sender] += msg.value;
    }

    function withdraw() {
        if (donations[msg.sender] > 0) {
            msg.sender.call.value(donations[msg.sender])();
            donations[msg.sender] = 0;
        }
    }
}
contract DAO {
    mapping (address => uint) donations;

    function donate() {
        donations[msg.sender] += msg.value;
    }

    function withdraw() {
        if (donations[msg.sender] > 0) {
            msg.sender.call.value(donations[msg.sender])();
            donations[msg.sender] = 0;
        }
    }
}
contract DAO {
  mapping (address => uint) donations;

  function donate() {
    donations[msg.sender] += msg.value;
  }

  function withdraw() {
    if (donations[msg.sender] > 0) {
      msg.sender.call.value(donations[msg.sender])();
      donations[msg.sender] = 0;
    }
  }
}
Formal semantics for EVM bytecode

- **Execution states:**
  \[(\mu, \iota, \sigma)_c\]
  - State of the local stack machine (pc, stack, memory, ...)
  - Execution environment (information to the internal transaction)
  - Global state (state of accounts on the blockchain)

- **Small-step relation:**
  \[\Gamma \vdash S \xrightarrow{a} S'\]
  - Transaction environment (information on the original transaction)
  - Call stacks of execution states

Annotation:
- Contract (address + code) that’s currently executed
- Call performed during execution (if any)
Reason for the DAO: untrusted contracts could influence the Ether flow of the contract (Call integrity)
Reason for the DAO: untrusted contracts could influence the Ether flow of the contract (Call integrity)
Reason for the DAO: untrusted contracts could influence the Ether flow of the contract (Call integrity)
Call integrity

- Reason for the DAO: untrusted contracts could influence the Ether flow of the contract (Call integrity)

Γ ⊨ s_c :: S \xrightarrow{\pi} t_c :: S' \land final(t_c) \land Γ ⊨ s'_c :: S' \xrightarrow{\pi'} t'_c :: S' \land final(t'_c)

\implies \pi \downarrow calls_c = \pi' \downarrow calls_{c'}
Call integrity

- Reason for the DAO: untrusted contracts could influence the Ether flow of the contract (Call integrity)

\[\Gamma \models s_c :: S \xrightarrow{\pi} t_c :: S \land \text{final}(t_c) \land \Gamma \models s'_c :: S' \xrightarrow{\pi'} t'_c :: S' \land \text{final}(t'_c) \]

\[\Rightarrow \pi \downarrow \text{calls}_c = \pi' \downarrow \text{calls}_c\]

Differing only in codes of untrusted addresses

=> c is called in the same way
Call integrity

- Reason for the DAO: untrusted contracts could influence the Ether flow of the contract (Call integrity)

\[ \Gamma \vdash s_c :: S \xrightarrow{\pi} t_c :: S \land \text{final}(t_c) \land \Gamma \vdash s'_c :: S' \xrightarrow{\pi'} t'_c :: S' \land \text{final}(t'_c) \]

\[ \Rightarrow \pi \downarrow \text{calls}_c = \pi' \downarrow \text{calls}_c \]

Differing only in codes of untrusted addresses

=> c is called in the same way

...
Single-Entrancy

- Single entrancy for c:
  “After being re-entered, contract c should not perform any more calls”
Single entrance for c: “After being re-entered, contract c should not perform any more calls”
Single-Entrancy

- Single entrancy for c: ”After being re-entered, contract c should not perform any more calls”

\[ \neg \exists s'' \in S, c' \in C \perp. \Gamma \models s_c :: S \rightarrow^* s_c'' :: s_c' :: S' + +s_c :: S \]
Single-Entrancy

- Single entrancy for c:
  “After being re-entered, contract c should not perform any more calls”

\[-\exists s'' \in S, c' \in C \perp. \Gamma \models s_c :: S \rightarrow^* \text{guard?} \rightarrow^* \text{invalid} \rightarrow^* \text{guard?} \rightarrow^* \text{guard?} \rightarrow^* \text{guard?} \rightarrow^* S' + +s_c :: S\]
Single entrancy for c: “After being re-entered, contract c should not perform any more calls”
Single-Entrancy

- Single entrancy for $c$:
  "After being re-entered, contract $c$ should not perform any more calls"

\[ \neg \exists s'' \in S, c' \in C \perp. \Gamma \models s_c :: S \rightarrow^{*} s''_c :: s'_c :: S' + +s_c :: S \]
Single-Entrancy

- Single entrancy for c:
  “After being re-entered, contract c should not perform any more calls”

\[ \neg \exists s'' \in S, c' \in C \perp. \Gamma \models s_c :: S \rightarrow^* s'' \quad s_c :: S' + +s_c :: S \]
Verifying Call integrity

Single-entrancy

“contract should not depend on return value of calls to untrusted contracts”

“contract should not depend on untrusted contract’s code (that it accesses directly)”

Call Integrity
Verifying Call integrity

- Single-entrancy
  - "contract should not depend on return value of calls to untrusted contracts"
- "contract should not depend on untrusted contract’s code (that it accesses directly)"

Hyper-Property

Call Integrity

Hard to verify
Verifying Call integrity

**Reachability Property**

- Single-entrancy

**Value-Dependency Properties**

- “contract should not depend on return value of calls to untrusted contracts”
- “contract should not depend on untrusted contract’s code (that it accesses directly)”

**Hyper-Property**

- Call Integrity

- Hard to verify
Verifying Call integrity

Reachability Property

Single-entrancy

Value-Dependency Properties

“contract should not depend on return value of calls to untrusted contract”

“contract should not depend on untrusted contract’s code (that it accesses directly)”

Verifiable using program dependence graphs

Call Integrity

Hyper-Property

Hard to verify
Verifying Call integrity

**Reachability Property**

- Single-entrancy
  - Ongoing work: Reachability analysis

**Value-Dependency Properties**

- “contract should not depend on return value of calls to untrusted contract”
- “contract should not depend on untrusted contract’s code (that it accesses directly)”

**Hyper-Property**

- Verifiable using program dependence graphs

**Call Integrity**

- Hard to verify
Towards static analysis

- Goal: Framework for checking reachability queries
- Approach:
  - Abstracted semantics as horn clause encoding
  - PDR (Property-directed reachability) engine of SMT-solver z3 for posing reachability queries
State abstraction

$mstate_{(id, pc)} (cd, (size, sa), aw)$

- Predicate parametrised by (artificial) contract id and pc
- Call depth
- Stack, represented as (unbounded) array + size
- Number of active words in memory

- Similar predicates for
  - Local memory
  - Global state
  - Execution environment
Horn clause encoding

- Execution steps modelled as horn clauses
  - horn clauses are generated according to the opcodes located at each pc
  - Example: Machine state rule for pc with opcode ADD
Execution steps modelled as horn clauses

- horn clauses are generated according to the opcodes located at each pc
- Example: Machine state rule for pc with opcode ADD

$$M_{\text{state}}(id, pc)(cd, (size, sa), aw)$$
$$\land size \geq 2$$
$$\land x = sa[size-1]$$
$$\land y = sa[size-2]$$
$$\Rightarrow M_{\text{state}}(id, pc+1)(cd, (size-1, sa[size-2 \rightarrow x+y]), aw)$$
Execution steps modelled as horn clauses
  • horn clauses are generated according to the opcodes located at each pc
  • Example: Machine state rule for pc with opcode ADD

\[
\begin{align*}
Mstate_{(id, pc)}(cd, (size, sa), aw) & \quad \land \quad \text{size} \geq 2 \\
& \quad \land \quad x = sa[\text{size}-1] \\
& \quad \land \quad y = sa[\text{size}-2] \\
\Rightarrow Mstate_{(id, pc+1)}(cd, (\text{size}-1, sa[\text{size}-2 \rightarrow x+y]), aw)
\end{align*}
\]

- simple range check
- stack is updated
- state predicate for next pc is implied
Horn clause encoding

- Execution steps modelled as horn clauses
  - horn clauses are generated according to the opcodes located at each pc
  - Example: Machine state rule for pc with opcode ADD

\[ \text{Mstate}_{(id, pc)}(cd, (size, sa), aw) \]
\[ \land \quad \text{size} \geq 2 \quad \text{(simple range check)} \]
\[ \land \quad x = sa[\text{size}-1] \]
\[ \land \quad y = sa[\text{size}-2] \]
\[ \implies \text{Mstate}_{(id, pc+1)}(cd, (\text{size}-1, sa[\text{size}-2 \rightarrow x+y]), aw) \]

\[ \text{Mstate}_{(id, pc)}(cd, (size, sa), aw) \]
\[ \implies \text{Exc}_{id}(cd) \]

state predicate for next pc is implied

stack is updated
Horn clause encoding

- Execution steps modelled as horn clauses
  - horn clauses are generated according to the opcodes located at each pc
  - Example: Machine state rule for pc with opcode \texttt{ADD}

\[
\text{Mstate}_{(id, pc)}(cd, (size, sa), aw) \\
\land \text{size} \geq 2 \quad \text{simple range check} \\
\land x = sa[\text{size-1}] \\
\land y = sa[\text{size-2}] \\
\Rightarrow \text{Mstate}_{(id, pc+1)}(cd, (size-1, sa[\text{size-2} \rightarrow x+y]), aw)
\]

\[
\Rightarrow \text{Exc}_{id}(cd) \\
\text{as we do not track gas explicitly, we assume the execution to stop in every step as it runs out of gas}
\]

state predicate for next pc is implied

stack is updated
Challenges

- For analysing a specific contract all its executions need to be approximated
For analysing a specific contract all its executions need to be approximated
For analysing a specific contract all its executions need to be approximated.
For analysing a specific contract all its executions need to be approximated
For analysing a specific contract all its executions need to be approximated.
For analysing a specific contract all its executions need to be approximated
For analysing a specific contract all its executions need to be approximated
For analysing a specific contract all its executions need to be approximated.
For analysing a specific contract all its executions need to be approximated
For analysing a specific contract all its executions need to be approximated.
For analysing a specific contract all its executions need to be approximated.

Challenges

- Tight approximation for call depth $x$
Challenges

- For analysing a specific contract all its executions need to be approximated.

Tight approximation for call depth $x$

Coarse approximation for call depth $y > x$

DAO contract

0xde2...

\begin{align*}
\vdots \\
guard? \\
\text{CALL} \\
\text{invalidate guard}
\end{align*}

\begin{align*}
x \\
x+2
\end{align*}

DAO contract

0xde2...

\begin{align*}
\vdots \\
guard? \\
\text{CALL} \\
\text{invalidate guard}
\end{align*}

\begin{align*}
? \\
x+2n
\end{align*}

DAO contract

0xde2...

\begin{align*}
\vdots \\
guard? \\
\text{CALL} \\
\text{invalidate guard}
\end{align*}

\begin{align*}
?
\end{align*}

Attacker contract

0xfc2...

\begin{align*}
? \\
\text{CALL}
\end{align*}
We prove our abstraction to be sound:

- “Every concrete execution can be mimicked by derivations in the abstract semantics”
Formal guarantees

- We prove our abstraction to be sound:
  - “Every concrete execution can be mimicked by derivations in the abstract semantics”

\[
\Gamma \models s_c :: S \rightarrow^* S' + + S \\
\Rightarrow \beta_s(s, id, 0) \cup \beta_C(C_{\text{call}}, id) \vdash \beta_S(S', C_{\text{call}}, id, 0)
\]
We prove our abstraction to be sound:

- “Every concrete execution can be mimicked by derivations in the abstract semantics”

\[
\Gamma \models s_c :: S \rightarrow^* S' + + S
\]

\[
\Rightarrow \beta_s(s, id, 0) \cup \beta_c(C_{\text{call}}, id) \models \beta_s(S', C_{\text{call}}, id, 0)
\]
We prove our abstraction to be sound:

• “Every concrete execution can be mimicked by derivations in the abstract semantics”

\[ \Gamma \models s_c :: S \rightarrow^* S' + + S \]

encoding of execution state as predicate instances

encoding of program logics of known contracts as horn clauses

encoding of call stack as predicate instances
We prove our abstraction to be sound:

- “Every concrete execution can be mimicked by derivations in the abstract semantics”

\[
\Gamma \models s_c :: S \Rightarrow S' + + S
\]

Non-reachability in the abstract setting carries over to the concrete setting
Security properties are independent of the concrete smart contract language

Different analysis techniques are required for verifying the security properties
  ● Value-dependency analysis (program dependence graphs)
  ● Trace property analysis

Future Work
  ● Smart contract synthesis
  ● Analysis of contract interactions
Questions?