Finding Permission Bugs in Smart Contracts with Role Mining

Ye Liu*, Yi Li*, Shang-Wei Lin*, Cyrille Artho+
*Nanyang Technological University
+KTH Royal Institute of Technology

Date: July 20, 2022
Smart Contract

- Manage valuable assets
- Involve multiple types of users with different capabilities
- Self-governed and once deployed, contract code cannot be changed

Enforcing access control correctly is crucial for smart contract implementations
Smart Contract

- Manage valuable assets
- Involve multiple types of users with different capabilities
- Self-governed and once deployed, contract code cannot be changed

Enforcing access control correctly is crucial for smart contract implementations

A decentralized finance application, ValueDeFi’s pool contract access control

- **Operator**: initializing the contract
- **Exchange proxy**: performing tasks on behalf of normal users
- **Fund agent**: allocating profits among normal users
- **Normal users**: depositing/withdrawing funds
Contents

- Smart contract and its access control
- Permission bug case study
- Our approach to find permission bugs
- Evaluation of the approach
- Conclusion
A permission bug in ValueDeFi

**Exploit:** On May 7, 2021, the contract ProfitSharingRewardPool, used by a Decentralized Finance (DeFi) platform named ValueDeFi, was hacked due to unprotected *initialize* function and lost around six million dollars.

```solidity
contract ProfitSharingRewardPool {
    ... 

    function initialize (address _stakeToken,
        address _liquidityToken,
        address _reserveFund) public notInitialized
    {
        stakeToken = _stakeToken;
        liquidityToken = _liquidityToken;
        reserveFund = _reserveFund;
        operator = msg.sender;
        setRewardPool(liquidityToken);
        + initialized = true  // bug-fix
    }
    ... 
}
```
A permission bug in ValueDeFi

**Exploit:** On May 7, 2021, the contract ProfitSharingRewardPool, used by a Decentralized Finance (DeFi) platform named ValueDeFi, was hacked due to unprotected `initialize` function and lost around six million dollars.

```solidity
contract ProfitSharingRewardPool {
    ...  
    function initialize (  
        address _stakeToken,
        address _liquidityToken,
        address _reserveFund) public notInitialized
    {
        stakeToken = _stakeToken;
        liquidityToken = _liquidityToken;
        reserveFund = _reserveFund;
        operator = msg.sender;
        setRewardPool(liquidityToken);
        +    initialized = true // bug-fix
    }
    ...  
}
```

(1) Pattern-based approach

Limitation:
- Static analysis: fail to realize that the `notInitialized` modifier will always return true, thus making the `initialize` function unprotected
- Dynamic analysis: lacking contract-specific test oracle on which type of user may invoke the initialize function

(2) Model-based approach

Limitation:
- Requiring customized model
Smart contract permission bug finding with role mining (SPCon)

Role structures

Access control policy

Bug finding

(u0, creation)
(u1, f1)
(u2, f2)
(u3, f3)
...
(un, fn)

Role mining

Transaction history

Smart contract

Conformance testing

Fig from Check by nareerat jaikaew from NounProject.com
Smart contract permission bug finding with role mining (SPCon)

- Extract user function access log from transaction history
Smart contract permission bug finding with role mining (SPCon)

- Role mining → infer user roles from existing user function access log.

![Diagram showing the process of role mining, access control policy, and bug finding.]
Smart contract permission bug finding with role mining (**SPCon**)

- Recover information-integrity access control policy from mined role structures
Smart contract permission bug finding with role mining (SPCon)

- **Conformance testing** → Check the conformance between contract implementation and its access control policy
Deploy smart contract to blockchain
Transaction history

Contract creation

Smart contract

blockchain

contract creator

contract creation
Transaction history

Smart contract

initialize

Contract creation

initialize
Transaction history

Smart contract

setExchangeProxy

Blockchain

Contract creation
- initialize
- setExchangeProxy
Transaction history

Smart contract

deposit

Blockchain

Contract creation
initialize
setExchangeProxy
deposit

Transaction history
Transaction history

B → deposit → Smart contract → blockchain

<table>
<thead>
<tr>
<th>Contract creation</th>
</tr>
</thead>
<tbody>
<tr>
<td>initialize</td>
</tr>
<tr>
<td>setExchangeProxy</td>
</tr>
<tr>
<td>deposit</td>
</tr>
<tr>
<td>deposit</td>
</tr>
</tbody>
</table>
Transaction history

Smart contract

 withdrew

Contract creation
- initialize
- setExchangeProxy
- deposit
- deposit
- withdraw
Transaction history

Smart contract

allocateMoreReward

blockchain

<table>
<thead>
<tr>
<th>Contract creation</th>
</tr>
</thead>
<tbody>
<tr>
<td>initialize</td>
</tr>
<tr>
<td>setExchangeProxy</td>
</tr>
<tr>
<td>deposit</td>
</tr>
<tr>
<td>deposit</td>
</tr>
<tr>
<td>withdraw</td>
</tr>
<tr>
<td>allocateMoreReward</td>
</tr>
</tbody>
</table>

Role structures

Access control policy

Bug finding

Transaction history
Transaction history

Smart contract

blockchain

Contract creation
- initialize
- setExchangeProxy
- deposit
- withdraw
- allocateMoreRewards
- deposit

D

deposit
Transaction history

Smart contract

withdraw

blockchain

<table>
<thead>
<tr>
<th>Contract creation</th>
</tr>
</thead>
<tbody>
<tr>
<td>initialize</td>
</tr>
<tr>
<td>setExchangeProxy</td>
</tr>
<tr>
<td>deposit</td>
</tr>
<tr>
<td>deposit</td>
</tr>
<tr>
<td>withdraw</td>
</tr>
<tr>
<td>allocateMoreRewards</td>
</tr>
<tr>
<td>deposit</td>
</tr>
<tr>
<td>withdraw</td>
</tr>
</tbody>
</table>
Transaction history

- Contract creation
  - initialize
  - setExchangeProxy
  - deposit
  - withdraw
  - allocateMoreRewards
- 
  - deposit
  - withdraw
  - 
  - depositFor
Transaction history

Smart contract

blockchain

Contract creation

initialize

setExchangeProxy

deposit

deposit

withdraw

allocateMoreRewards

deposit

withdraw

depositFor

deposit
User access log

<table>
<thead>
<tr>
<th>User access matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td><img src="image" alt="User access matrix" /></td>
</tr>
</tbody>
</table>

**Contract creation**
- initialize
- setExchangeProxy
- deposit
- withdraw
- allocateMoreReward
- deposit
- withdraw
- depositFor
- deposit
User access log is Incomplete

<table>
<thead>
<tr>
<th>User access matrix is Partial</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Contract creation</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>initialize</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>setExchangeProxy</td>
<td></td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>deposit</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>deposit</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>withdraw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>allocateMoreReward</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>deposit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>withdraw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>depositFor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>deposit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
</tbody>
</table>

20
Role mining from partial observation

- **Assumption:** Users are likely to belong to the same role if they have
  (a) accessed the exact same set of functions.
  (b) called common set of functions with similar usage frequency.

- **Challenges**
  ➢ Considering only (a) would create too many spurious roles
  ➢ However, considering (b) can lead to many mismatches with the observation.
  ➢ NP-hard problem.
Role mining from partial observation

- Genetic algorithm solution
  (a) Frequency similarity metric: measure the chance of a spurious role.
  (b) Consistency metric: measure the mismatch with the observation.

- Role mining steps
  ➢ Infer basic roles: Group users having the same set of function calls
  ➢ Merge basic roles: Combine those with similar frequency patterns
Role mining result

<table>
<thead>
<tr>
<th>Role</th>
<th>Contract creation</th>
</tr>
</thead>
<tbody>
<tr>
<td>initialize</td>
<td></td>
</tr>
<tr>
<td>setExchangeProxy</td>
<td></td>
</tr>
<tr>
<td>deposit</td>
<td></td>
</tr>
<tr>
<td>withdraw</td>
<td></td>
</tr>
<tr>
<td>allocateMoreRewards</td>
<td></td>
</tr>
<tr>
<td>deposit</td>
<td></td>
</tr>
<tr>
<td>withdraw</td>
<td></td>
</tr>
<tr>
<td>depositFor</td>
<td></td>
</tr>
</tbody>
</table>
## Role structure

<table>
<thead>
<tr>
<th>Users (UA)</th>
<th>Permissions to Functions (PA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ Operator }</td>
<td>{ initialize(), setOperator(), setExchangeProxy(), setReserveFund(), depositFor(), allocateMoreRewards() }</td>
</tr>
<tr>
<td>{ Exchange proxy }</td>
<td>{ setExchangeProxy(), depositFor() }</td>
</tr>
<tr>
<td>{ Fund agent }</td>
<td>{ setReserveFund(), allocateMoreRewards() }</td>
</tr>
<tr>
<td>{ Normal Users }</td>
<td>{ deposit(), withdraw(), claimRewards() }</td>
</tr>
</tbody>
</table>
## Role structure

<table>
<thead>
<tr>
<th>Users (UA)</th>
<th>Permissions to Functions (PA)</th>
<th>Written State Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ Operator }</td>
<td>{ initialize(), setOperator(), setExchangeProxy(), setReserveFund(), depositFor(), allocateMoreRewards() }</td>
<td>{operator, stakeToken, liquidityTo-ken, exchangeProxy, reserveFund}</td>
</tr>
<tr>
<td>{ Exchange proxy }</td>
<td>{ setExchangeProxy(), depositFor() }</td>
<td>{exchangeProxy}</td>
</tr>
<tr>
<td>{ Fund agent }</td>
<td>{ setReserveFund(), allocateMoreRewards() }</td>
<td>{reserveFund}</td>
</tr>
<tr>
<td>{ Normal Users }</td>
<td>{ deposit(), withdraw(), claimRewards() }</td>
<td>{}</td>
</tr>
</tbody>
</table>
Access control policy (information security lattice)

Operator
Write: {operator, stakeToken, liquidityToken, exchangeProxy, reserveFund}

Exchange proxy
write: {exchangeProxy}

Fund agent
Write: {reserveFund}

Normal Users
Write: {}
Conformance testing

- Symbolic execution.
- Concrete value from blockchain snapshot.

**Test Sequences**

<table>
<thead>
<tr>
<th>Test Sequence</th>
<th>Policy check</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{ts1} = \text{setExchangeProxy()} \rightarrow \text{depositFor()}$</td>
<td>Safe</td>
</tr>
<tr>
<td>$\text{ts2} = \text{setReservedFund()} \rightarrow \text{allocateMoreRewards()}$</td>
<td>Safe</td>
</tr>
<tr>
<td>$\text{ts3} = \text{deposit}(X) \rightarrow \text{withdraw}(Y)$</td>
<td>Safe</td>
</tr>
<tr>
<td>$\text{ts4} = \text{initialize()} \rightarrow \text{setExchangeProxy()}$</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>
Conformance testing

- Symbolic execution.
- Concrete value from blockchain snapshot.

<table>
<thead>
<tr>
<th>Test Sequences</th>
<th>Policy check</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ts1 = setExchangeProxy(_) -&gt; depositFor(_)</code></td>
<td>Safe</td>
</tr>
<tr>
<td><code>ts2 = setReservedFund() -&gt; allocateMoreRewards()</code></td>
<td>Safe</td>
</tr>
<tr>
<td><code>ts3 = deposit(X) -&gt; withdraw(Y)</code></td>
<td>Safe</td>
</tr>
<tr>
<td><code>ts4 = initialize() -&gt; setExchangeProxy()</code></td>
<td>Unsafe</td>
</tr>
</tbody>
</table>
Conformance testing

- Symbolic execution.
- Concrete value from blockchain snapshot.

<table>
<thead>
<tr>
<th>Test Sequences</th>
<th>Policy check</th>
</tr>
</thead>
<tbody>
<tr>
<td>ts1 = setExchangeProxy(<em>) -&gt; depositFor(</em>)</td>
<td>Safe</td>
</tr>
<tr>
<td>ts2 = setReservedFund() -&gt; allocateMoreRewards()</td>
<td>Safe</td>
</tr>
<tr>
<td>ts3 = deposit(X) -&gt; withdraw(Y)</td>
<td>Safe</td>
</tr>
<tr>
<td>ts4 = initialize() -&gt; setExchangeProxy()</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

### Test Sequences

- **ts1 = setExchangeProxy(_) -> depositFor(_)**: Safe
- **ts2 = setReservedFund() -> allocateMoreRewards()**: Safe
- **ts3 = deposit(X) -> withdraw(Y)**: Safe
- **ts4 = initialize() -> setExchangeProxy()**: Unsafe
Conformance testing

- Symbolic execution.
- Concrete value from blockchain snapshot.

<table>
<thead>
<tr>
<th>Test Sequences</th>
<th>Policy check</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{ts1} = \text{setExchangeProxy}() \rightarrow \text{depositFor}() )</td>
<td>Safe</td>
</tr>
<tr>
<td>( \text{ts2} = \text{setReservedFund}() \rightarrow \text{allocateMoreRewards}() )</td>
<td>Safe</td>
</tr>
<tr>
<td>( \text{ts3} = \text{deposit}(X) \rightarrow \text{withdraw}(Y) )</td>
<td>Safe</td>
</tr>
<tr>
<td>( \text{ts4} = \text{initialize}() \rightarrow \text{setExchangeProxy}() )</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

Symbolic execution.
Concrete value from blockchain snapshot.
Conformance testing

- Symbolic execution.
- Concrete value from blockchain snapshot.

Test Sequences | Policy check
---|---
\( ts_1 = \text{setExchangeProxy}(\_ \rightarrow \text{depositFor}(\_)) \) | Safe
\( ts_2 = \text{setReservedFund}() \rightarrow \text{allocateMoreRewards}() \) | Safe
\( ts_3 = \text{deposit}(X) \rightarrow \text{withdraw}(Y) \) | Safe
\( ts_4 = \text{initialize}() \rightarrow \text{setExchangeProxy}() \) | Unsafe
Conformance testing

- Symbolic execution.
- Concrete value from blockchain snapshot.

<table>
<thead>
<tr>
<th>Test Sequences</th>
<th>Policy check</th>
</tr>
</thead>
<tbody>
<tr>
<td>ts1 = setExchangeProxy(<em>) -&gt; depositFor(</em>)</td>
<td>Safe</td>
</tr>
<tr>
<td>ts2 = setReservedFund() -&gt; allocateMoreRewards()</td>
<td>Safe</td>
</tr>
<tr>
<td>ts3 = deposit(X) -&gt; withdraw(Y)</td>
<td>Safe</td>
</tr>
<tr>
<td>ts4 = initialize() -&gt; setExchangeProxy()</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>
Conformance testing

- Symbolic execution.
- Concrete value from blockchain snapshot.

<table>
<thead>
<tr>
<th>Test Sequences</th>
<th>Policy check</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1 = \text{setExchangeProxy}() \rightarrow \text{depositFor}()$</td>
<td>Safe</td>
</tr>
<tr>
<td>$t_2 = \text{setReservedFund}() \rightarrow \text{allocateMoreRewards}()$</td>
<td>Safe</td>
</tr>
<tr>
<td>$t_3 = \text{deposit}(X) \rightarrow \text{withdraw}(Y)$</td>
<td>Safe</td>
</tr>
<tr>
<td>$t_4 = \text{initialize}() \rightarrow \text{setExchangeProxy}()$</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>
Conformance testing

- Symbolic execution.
- Concrete value from blockchain snapshot.

<table>
<thead>
<tr>
<th>Test Sequences</th>
<th>Policy check</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ts1 = \text{setExchangeProxy}(<em>) \rightarrow \text{depositFor}(</em>)$</td>
<td>Safe</td>
</tr>
<tr>
<td>$ts2 = \text{setReservedFund}() \rightarrow \text{allocateMoreRewards}()$</td>
<td>Safe</td>
</tr>
<tr>
<td>$ts3 = \text{deposit}(X) \rightarrow \text{withdraw}(Y)$</td>
<td>Safe</td>
</tr>
<tr>
<td>$ts4 = \text{initialize}() \rightarrow \text{setExchangeProxy}()$</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

Test Sequences

- $ts1 = \text{setExchangeProxy}(_) \rightarrow \text{depositFor}(_)$
  - Policy check: Safe
- $ts2 = \text{setReservedFund}() \rightarrow \text{allocateMoreRewards}()$
  - Policy check: Safe
- $ts3 = \text{deposit}(X) \rightarrow \text{withdraw}(Y)$
  - Policy check: Safe
- $ts4 = \text{initialize}() \rightarrow \text{setExchangeProxy}()$
  - Policy check: Unsafe

Symbolic execution.
Concrete value from blockchain snapshot.
Conformance testing

- Symbolic execution.
- Concrete value from blockchain snapshot.

<table>
<thead>
<tr>
<th>Test Sequences</th>
<th>Policy check</th>
</tr>
</thead>
<tbody>
<tr>
<td>ts1 = setExchangeProxy(<em>) -&gt; depositFor(</em>)</td>
<td>Safe</td>
</tr>
<tr>
<td>ts2 = setReservedFund() -&gt; allocateMoreRewards()</td>
<td>Safe</td>
</tr>
<tr>
<td>ts3 = deposit(X) -&gt; withdraw(Y)</td>
<td>Safe</td>
</tr>
<tr>
<td>ts4 = initialize() -&gt; setExchangeProxy()</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

Symbolic execution.
Concrete value from blockchain snapshot.

Test Sequences:
- **ts1**: setExchangeProxy(_) -> depositFor(_)
- **ts2**: setReservedFund() -> allocateMoreRewards()
- **ts3**: deposit(X) -> withdraw(Y)
- **ts4**: initialize() -> setExchangeProxy()
Conformance testing

- Symbolic execution.
- Concrete value from blockchain snapshot.

<table>
<thead>
<tr>
<th>Test Sequences</th>
<th>Policy check</th>
</tr>
</thead>
<tbody>
<tr>
<td>ts1 = setExchangeProxy(<em>) -&gt; depositFor(</em>)</td>
<td>Safe</td>
</tr>
<tr>
<td>ts2 = setReservedFund() -&gt; allocateMoreRewards()</td>
<td>Safe</td>
</tr>
<tr>
<td>ts3 = deposit(X) -&gt; withdraw(Y)</td>
<td>Safe</td>
</tr>
<tr>
<td>ts4 = initialize() -&gt; setExchangeProxy()</td>
<td>Unsafe</td>
</tr>
</tbody>
</table>

**ts4 is an exploit attack sequence to the permission bug and we generate concrete transactions as the PoCs.**
Evaluation

- Answering 3 Research Questions:

  **Accuracy and efficiency of role mining**
  RQ1: How accurately and efficiently does SPCon learn the user roles?

  **Performance in permission bug finding**
  RQ2: How does SPCon perform in detecting permission bugs?

  **Discussion**
  RQ3: Why do existing tools fail to detect many permission bugs, how does our approach improve on this?
RQ1: Accuracy and efficiency of role mining

<table>
<thead>
<tr>
<th>Approach</th>
<th>Run time (s)</th>
<th>Number of roles</th>
<th>Number of mined roles per roles in ground truth</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPr</td>
<td>0.21</td>
<td>8.28</td>
<td>2.69</td>
</tr>
<tr>
<td>ORCA</td>
<td>5.08</td>
<td>21.96</td>
<td>7.17</td>
</tr>
<tr>
<td>HM</td>
<td>49.54</td>
<td>19.04</td>
<td>6.37</td>
</tr>
<tr>
<td>GO</td>
<td>191.72</td>
<td>15.34</td>
<td>4.86</td>
</tr>
<tr>
<td>SPCon (0.4, 0.6)</td>
<td>31.69</td>
<td>7.00</td>
<td>2.27</td>
</tr>
<tr>
<td>SPCon (0.5, 0.5)</td>
<td>33.10</td>
<td>4.64</td>
<td>1.54</td>
</tr>
<tr>
<td>SPCon (0.6, 0.4)</td>
<td>34.55</td>
<td>3.52</td>
<td>1.11</td>
</tr>
</tbody>
</table>

SPCon can accurately and efficiently reverse engineer likely user roles of smart contracts
RQ2: Performance in permission bug finding on the benchmark SmartBugs.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Number of vulnerabilities</th>
<th>Agress (&gt;=1) Num (%)</th>
<th>True-positive rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slither</td>
<td>2,356</td>
<td>568 (24%)</td>
<td>24.2%</td>
</tr>
<tr>
<td>Securify</td>
<td>614</td>
<td>93 (15%)</td>
<td>32.8%</td>
</tr>
<tr>
<td>SmartCheck</td>
<td>384</td>
<td>193 (50%)</td>
<td>29.3%</td>
</tr>
<tr>
<td>Mythril</td>
<td>1076</td>
<td>460 (43%)</td>
<td>39.0%</td>
</tr>
<tr>
<td>Maian</td>
<td>44</td>
<td>29 (66%)</td>
<td>61.4%</td>
</tr>
<tr>
<td>Manticore</td>
<td>47</td>
<td>19 (40%)</td>
<td>19.1%</td>
</tr>
<tr>
<td>SPCon</td>
<td>44</td>
<td>33 (75%)</td>
<td>81.8%</td>
</tr>
</tbody>
</table>

SPCon shows higher true positive rate (81.8%) compared to the existing tools. Moreover, SPCon found 11 previously unknown permission bugs in the SmartBugs benchmark.
RQ2: Performance in permission bug finding on 17 permission CVEs.

SPCon can find more permission CVEs (nine) compared to other existing tools.
RQ3: Discussion

Why the current tools fail to detect some permission bugs?
• Limited and overly generic, "one size fits all" approach. They only cover some kinds of permission bugs, e.g., the use of modifier

How does our approach improve on this?
• Learn access control model tailored to each contract
Conclusion

Our main contributions include:

1. Learn permission model from transaction history.
2. Generate executable exploits.

Smart contract → Transaction history → Role mining → Role structures → Access control policy → Bug finding

- (u0, creation)
- (u1, f1)
- (u2, f2)
- (u3, f3)
- ...
- (un, fn)