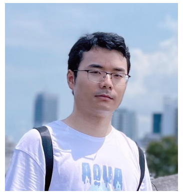
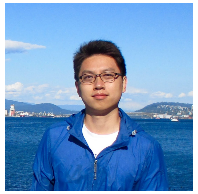
# Finding Permission Bugs in Smart Contracts with Role Mining

**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

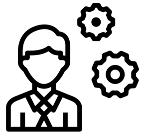
**Enforcing access control correctly is crucial for smart contract implementations**

# Smart Contract

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

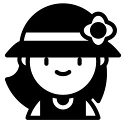
**Enforcing access control correctly is crucial for smart contract implementations**

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

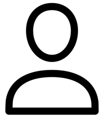


Operator: initializing the contract

Exchange proxy: performing tasks on behalf of normal users



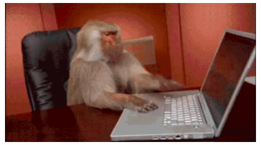
Fund agent: allocating profits among normal users



Normal users: depositing/ withdrawing funds

# Contents

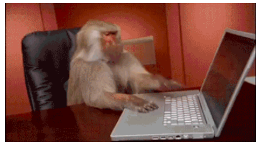
# A permission bug in ValueDeFi



**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

```
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

```
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*)

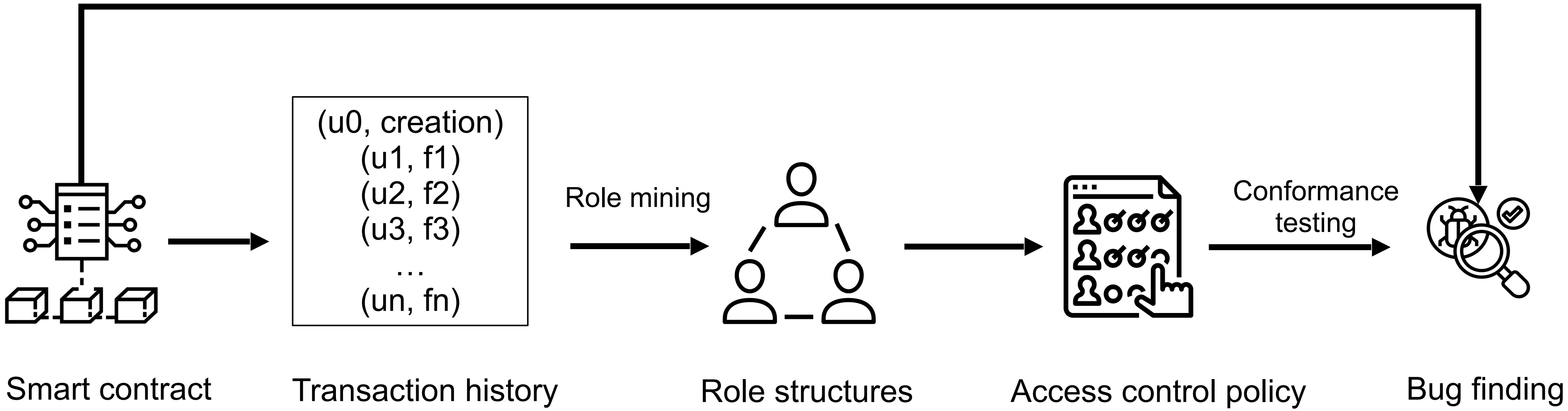



Fig from Check by nareerat jaikaew from NounProject.com

# Smart contract permission bug finding with role mining (*SPCon*)

- Extract user function access log from transaction history

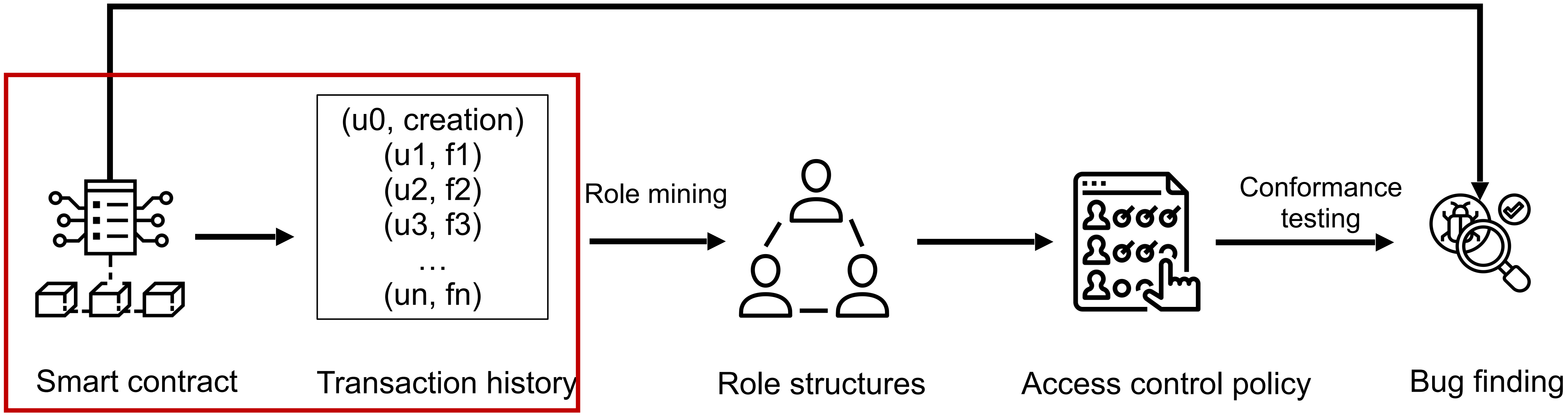



Fig from Check by nareerat jaikaew from NounProject.com

# Smart contract permission bug finding with role mining (*SPCon*)

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

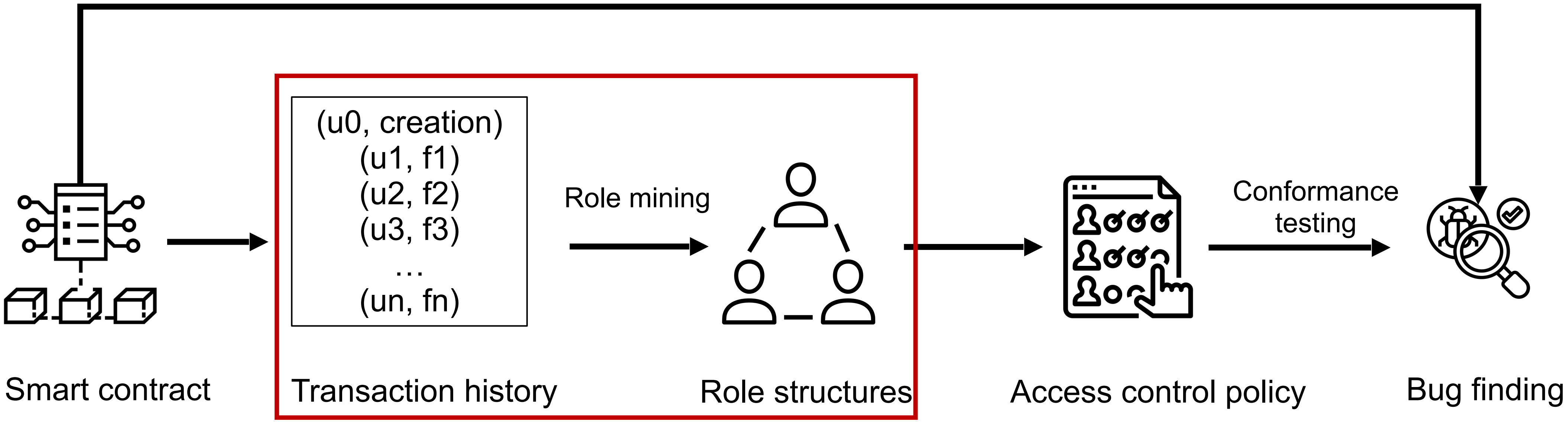



Fig from Check by nareerat jaikaew from NounProject.com

# Smart contract permission bug finding with role mining (*SPCon*)

- Recover information-integrity access control policy from mined role structures

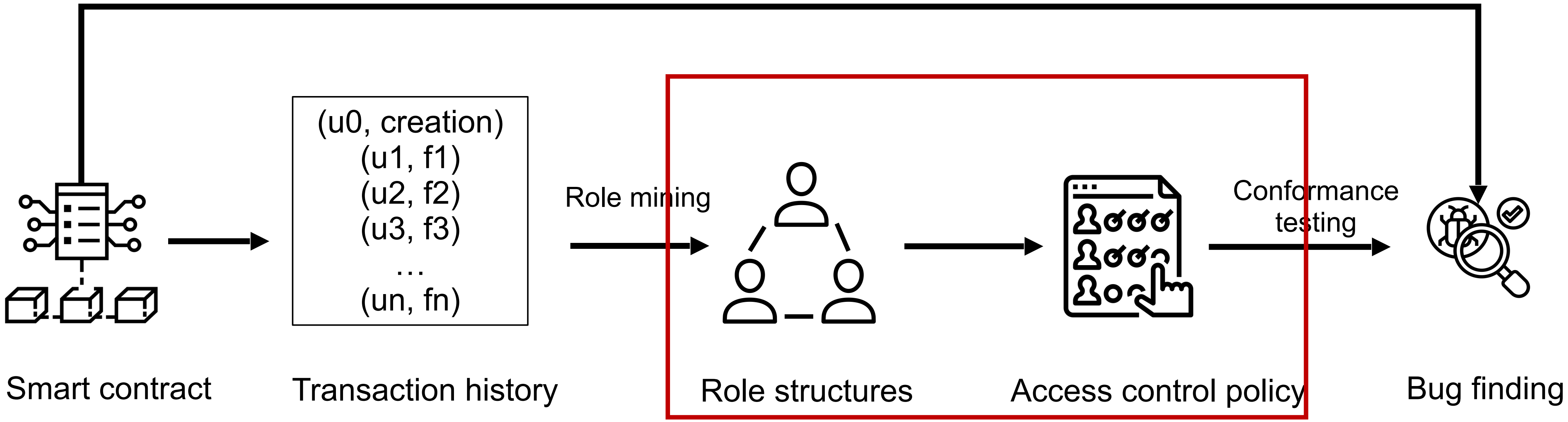



Fig from Check by nareerat jaikaew from NounProject.com

# Smart contract permission bug finding with role mining (*SPCon*)

- **Conformance testing →** Check the conformance between contract implementation and its access control policy

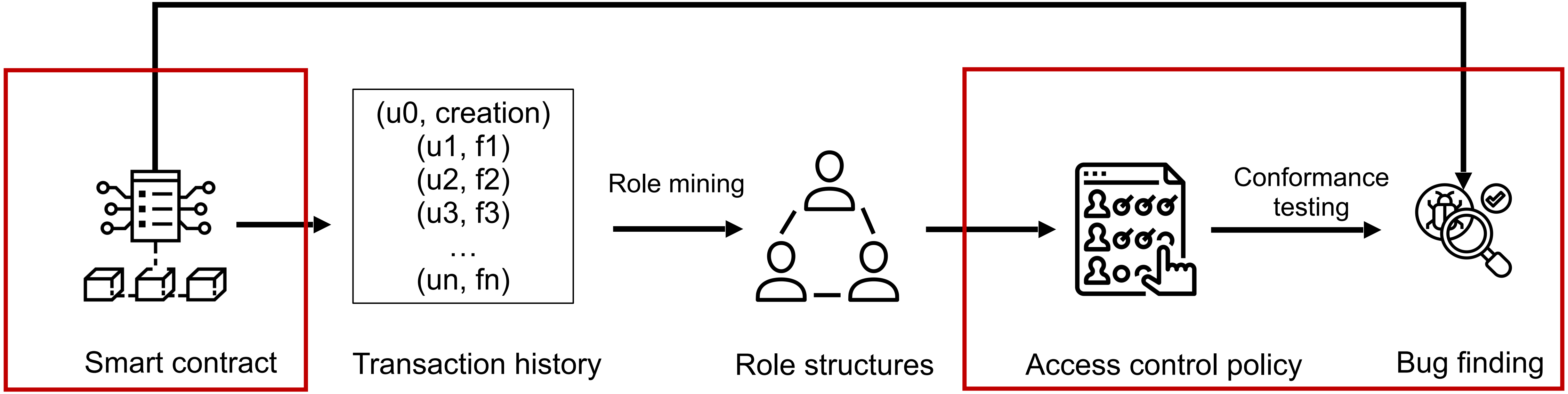
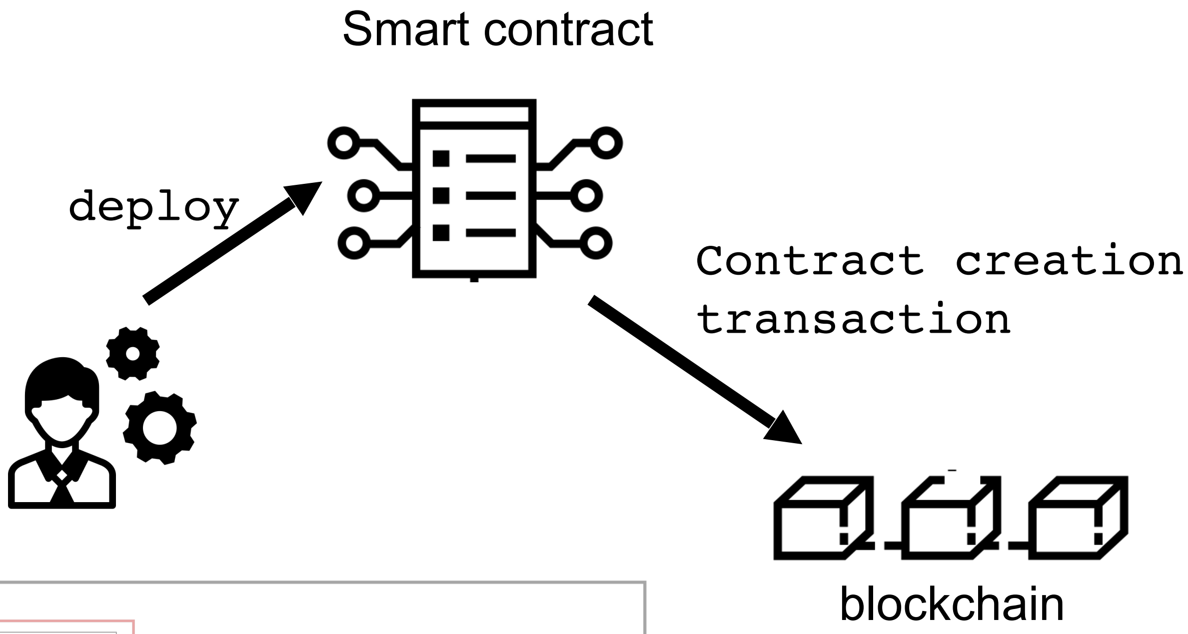
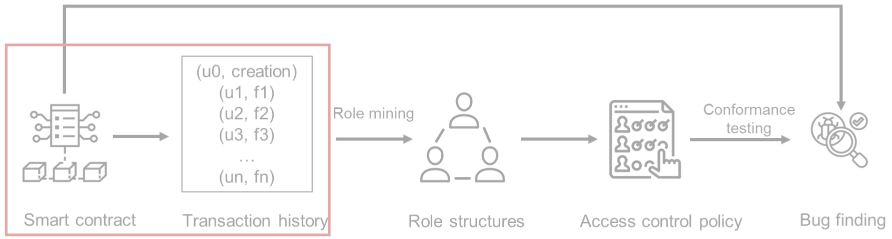


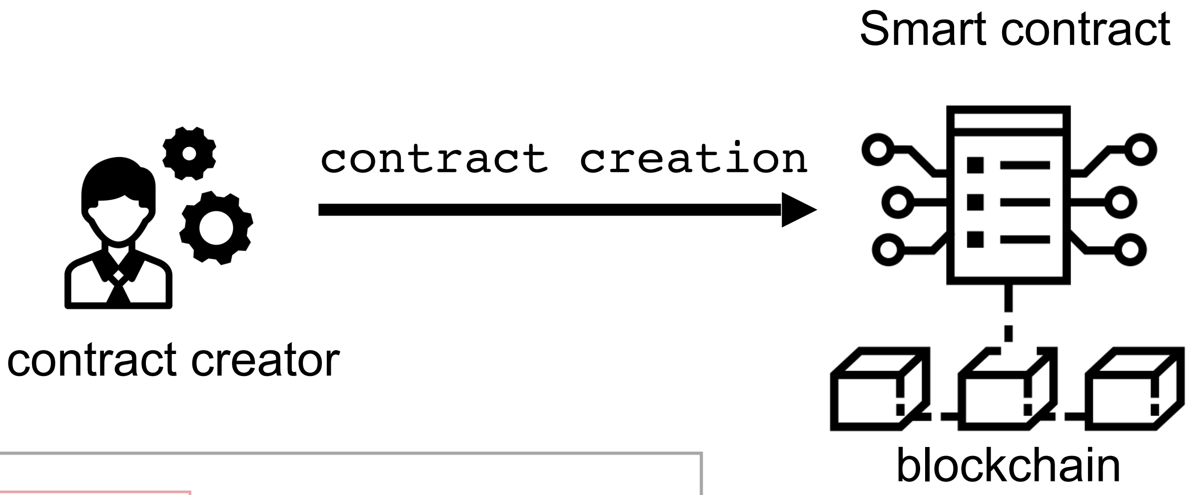

Fig from Check by nareerat jaikaew from NounProject.com

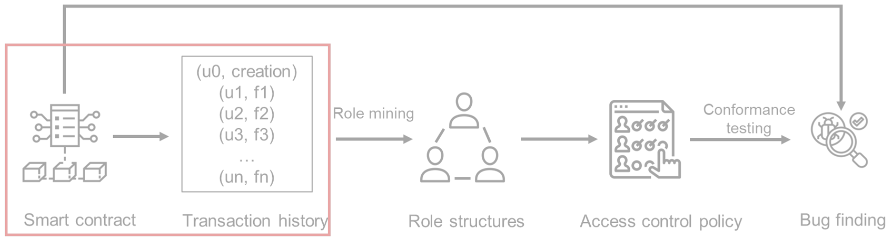
# Deploy smart contract to blockchain



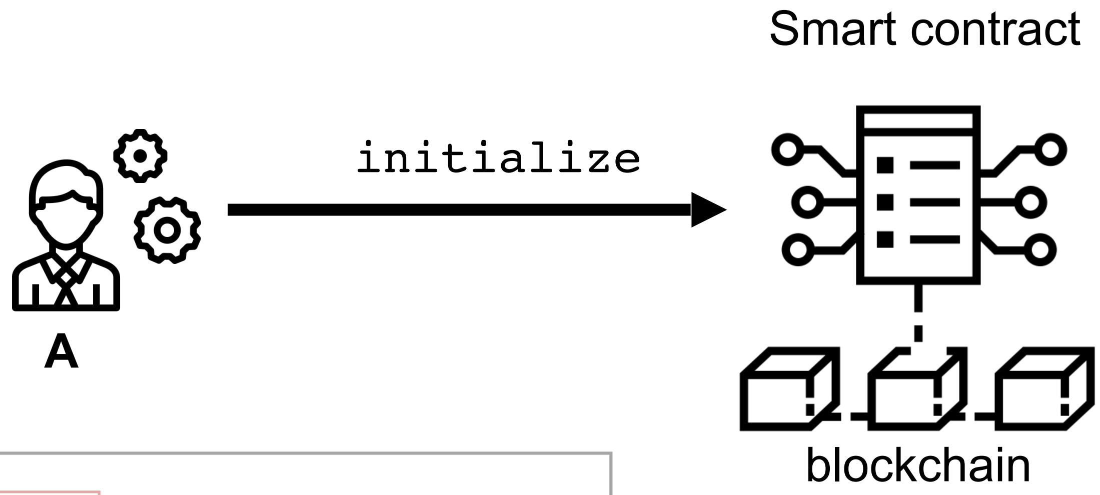
# Transaction history



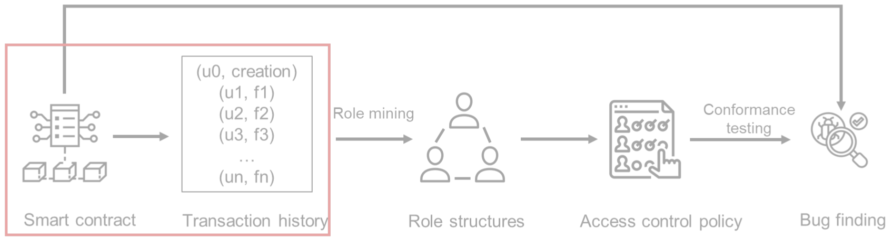




Contract creation

# Transaction history

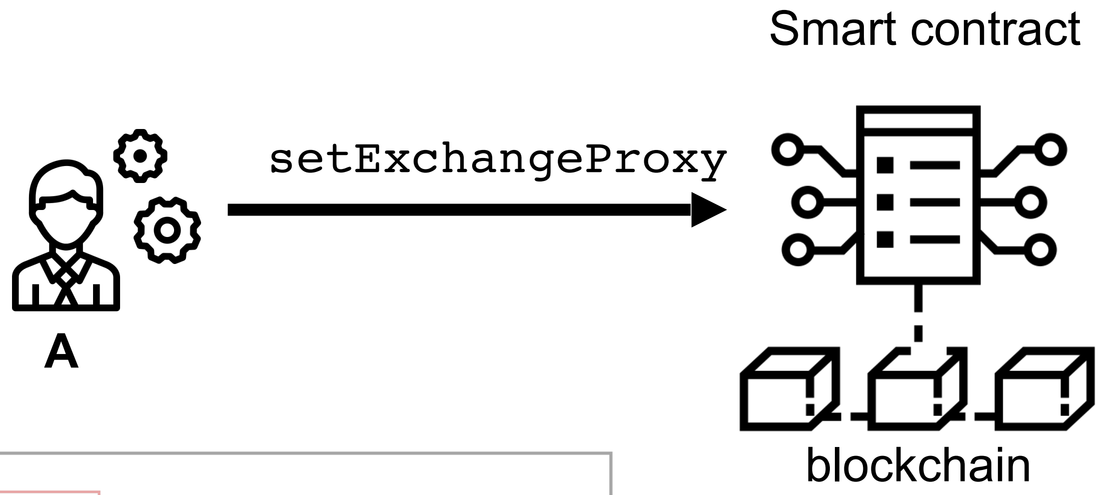



| | |
|---|---|
| | Contract creation |
| | `initialize` |

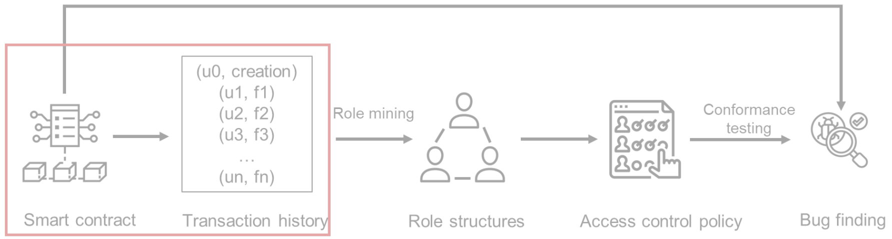


NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

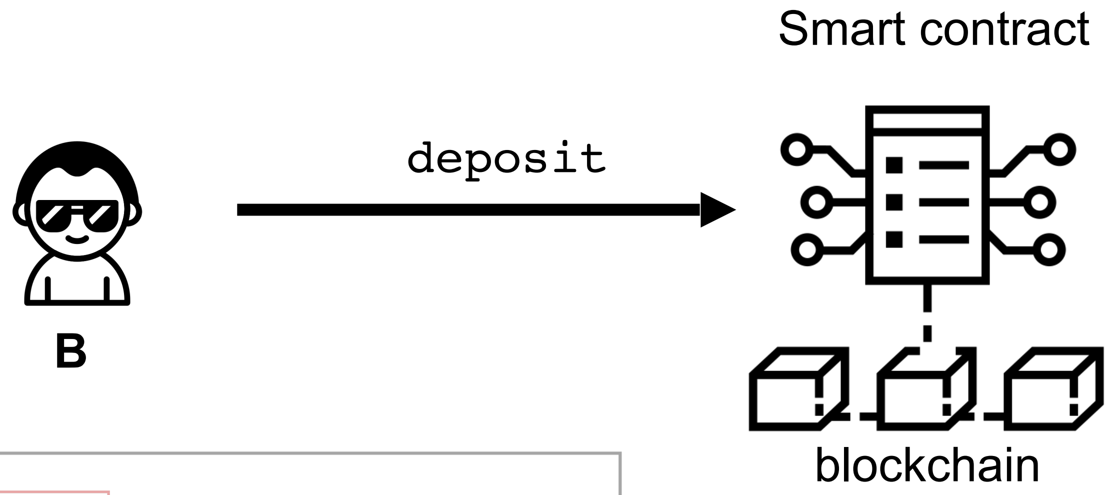
# Transaction history



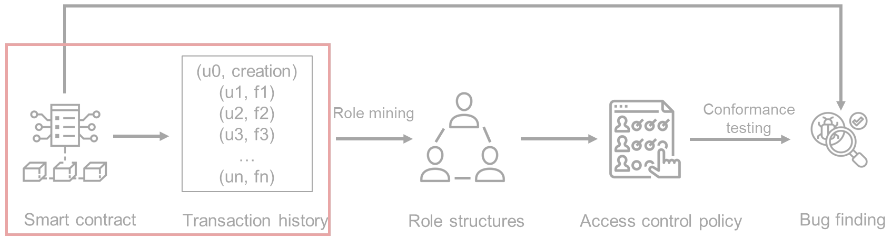

| | |
|---|---|
| | Contract creation |
| | `initialize` |
| | **`setExchangeProxy`** |



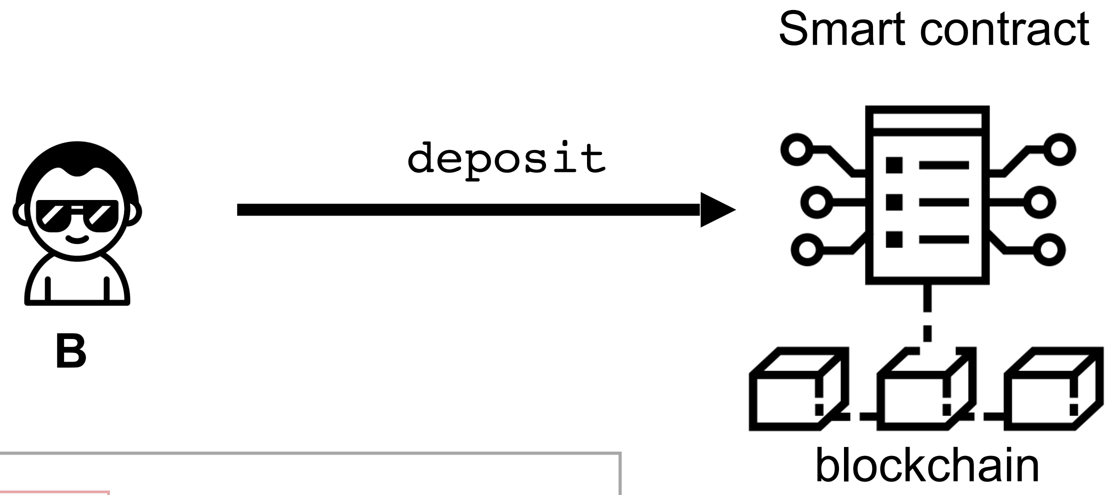
# Transaction history



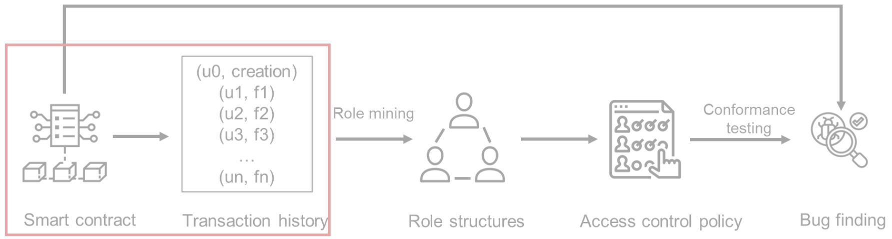

| | |
|---|---|
| | Contract creation |
| | `initialize` |
| | `setExchangeProxy` |
| | **`deposit`** |



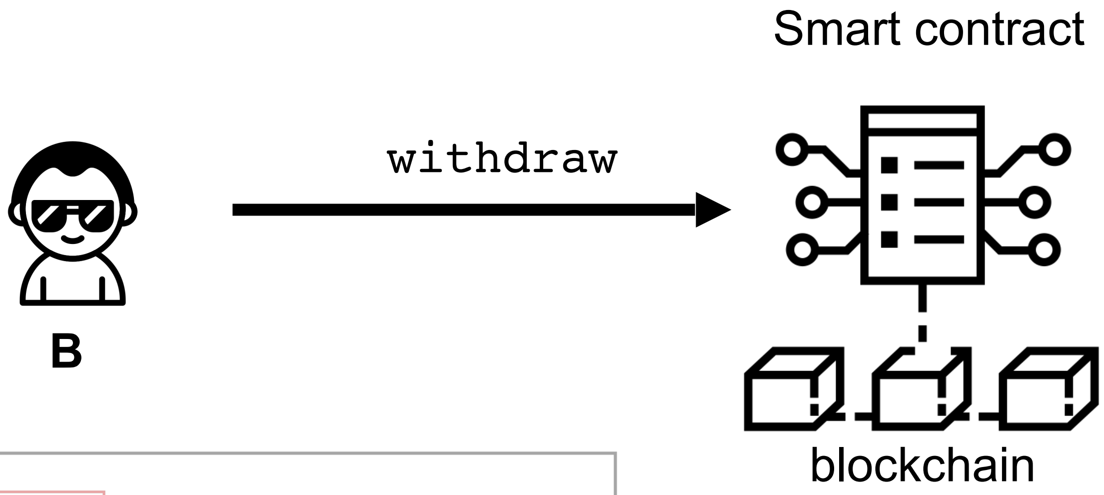
# Transaction history



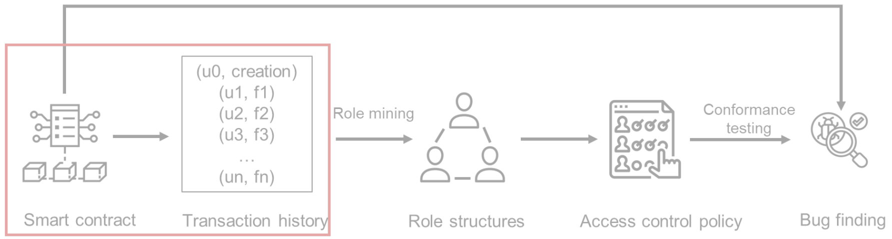

| | |
|---|---|
| | Contract creation |
| | `initialize` |
| | `setExchangeProxy` |
| | `deposit` |
| | **`deposit`** |



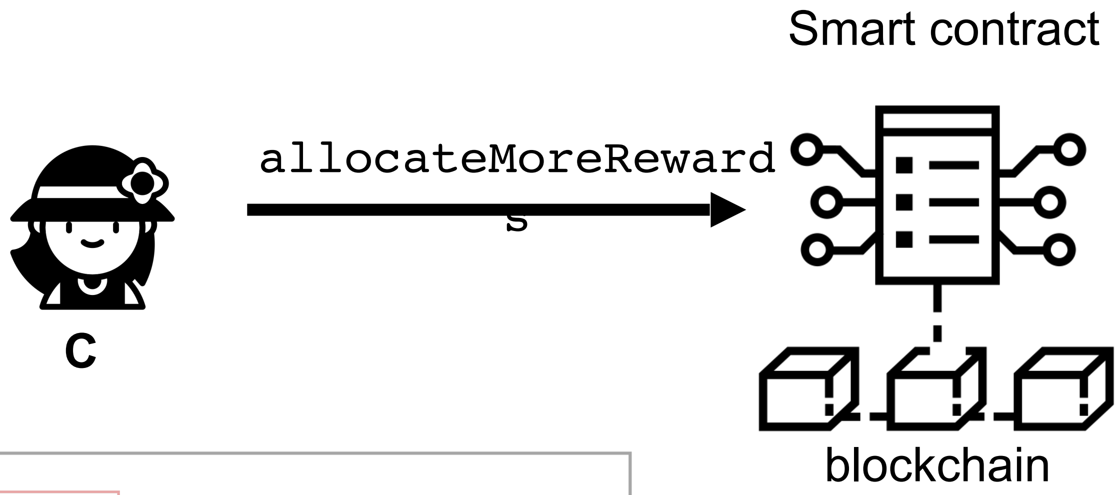
# Transaction history



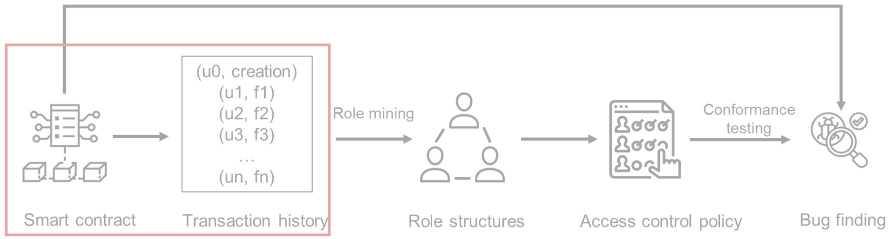

| | |
|---|---|
| | Contract creation |
| | initialize |
| | setExchangeProxy |
| | deposit |
| | deposit |
| | **withdraw** |



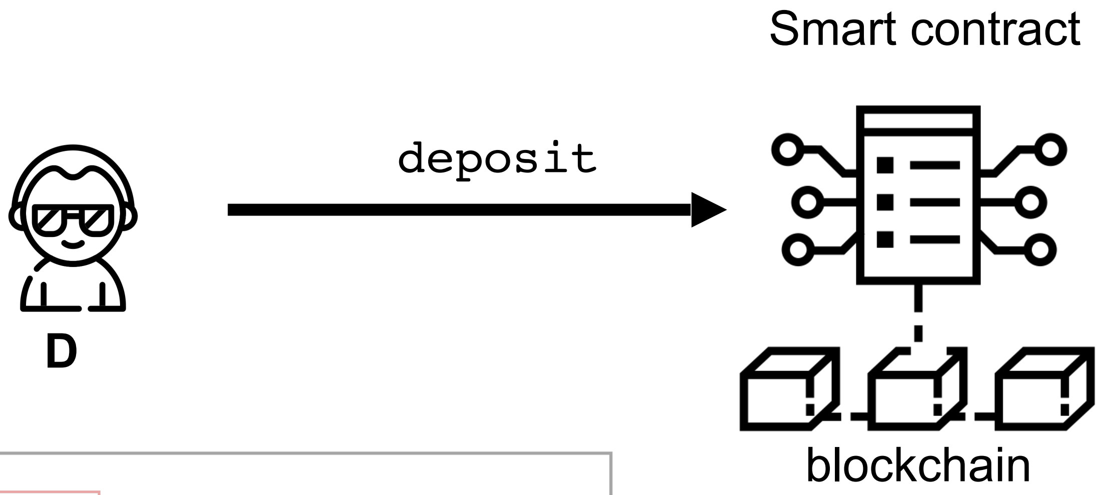
# Transaction history

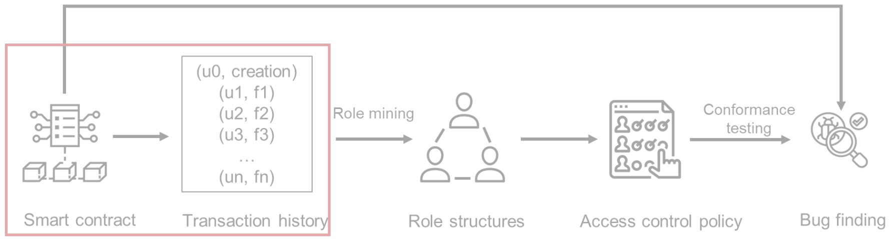



| | |
|---|---|
| | Contract creation |
| | `initialize` |
| | `setExchangeProxy` |
| | `deposit` |
| | `deposit` |
| | withdraw |
| | **`allocateMoreRewards`** |



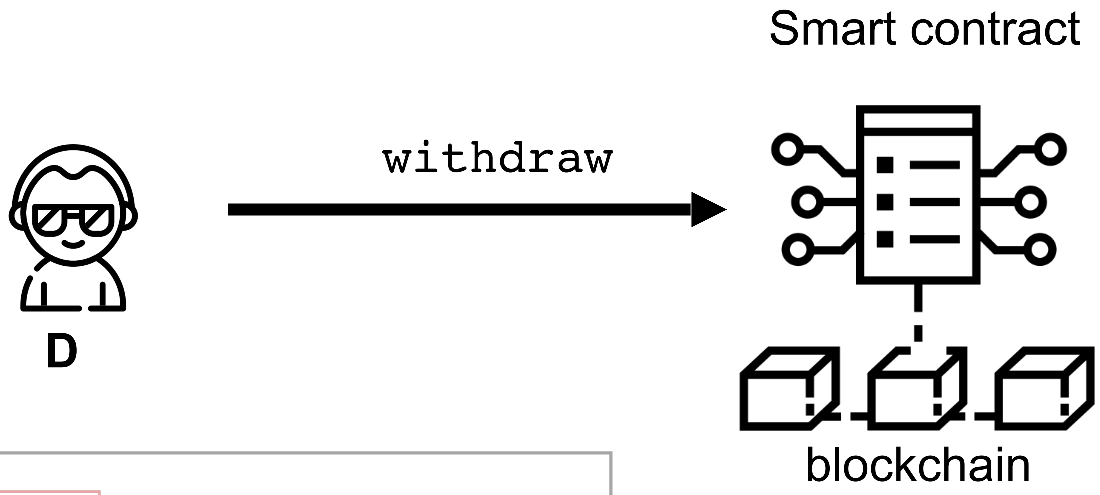
# Transaction history

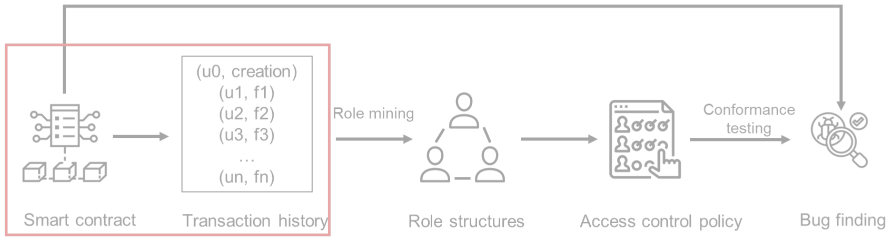



| | |
|---|---|
| | Contract creation |
| | initialize |
| | setExchangeProxy |
| | deposit |
| | deposit |
| | withdraw |
| | allocateMoreRewards |
| | **deposit** |

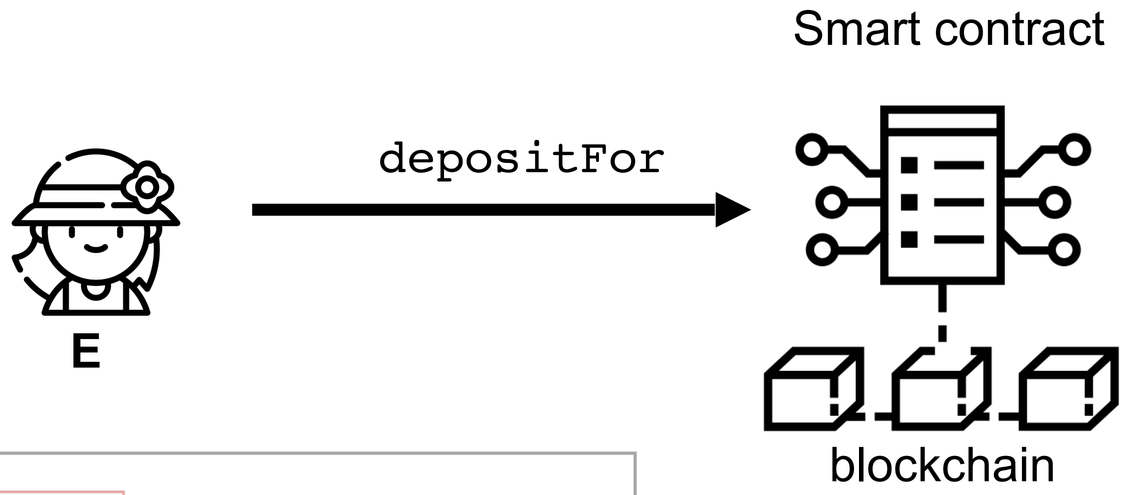
# Transaction history



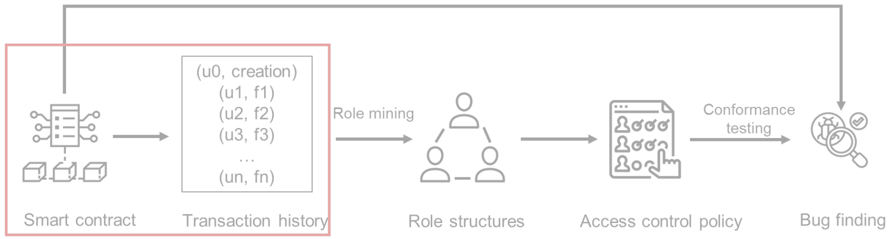




| |
|---|
| Contract creation |
| `initialize` |
| `setExchangeProxy` |
| `deposit` |
| `deposit` |
| withdraw |
| `allocateMoreRewards` |
| `deposit` |
| **withdraw** |

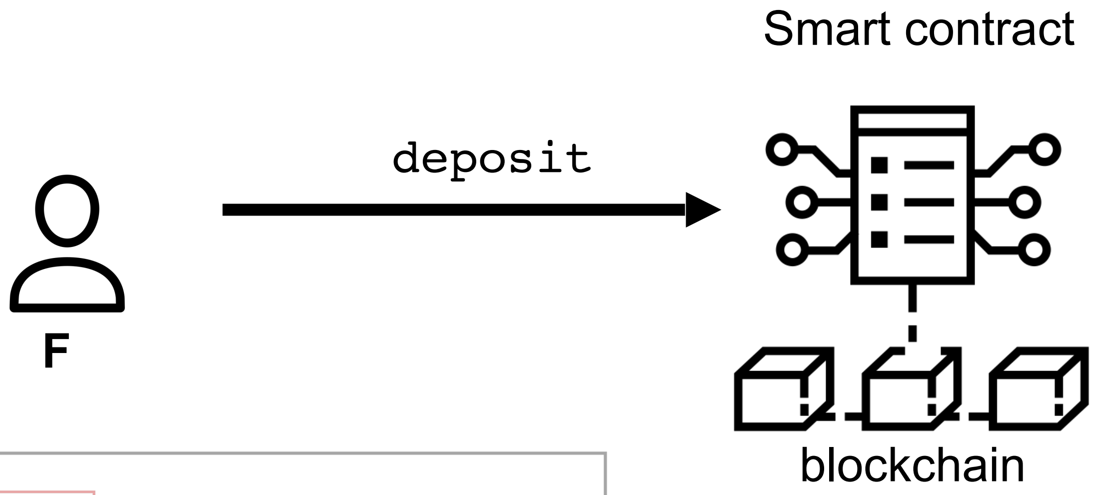
# Transaction history

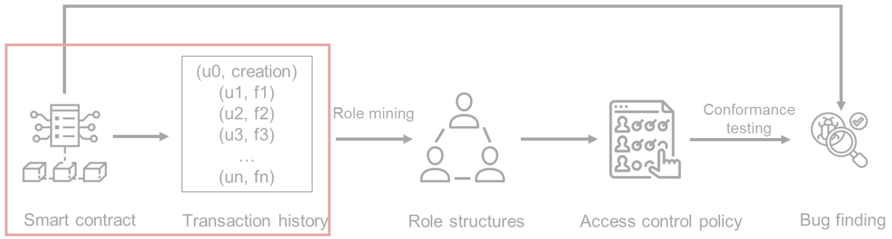



| |
|---|
| Contract creation |
| initialize |
| setExchangeProxy |
| deposit |
| deposit |
| withdraw |
| allocateMoreRewards |
| deposit |
| withdraw |
| **depositFor** |

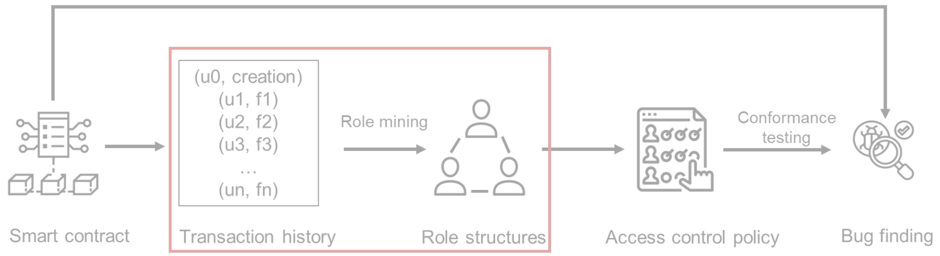


# Transaction history




| | |
|---|---|
| | Contract creation |
| | initialize |
| | setExchangeProxy |
| | deposit |
| | deposit |
| | withdraw |
| | allocateMoreRewards |
| | deposit |
| | withdraw |
| | depositFor |
| | **deposit** |

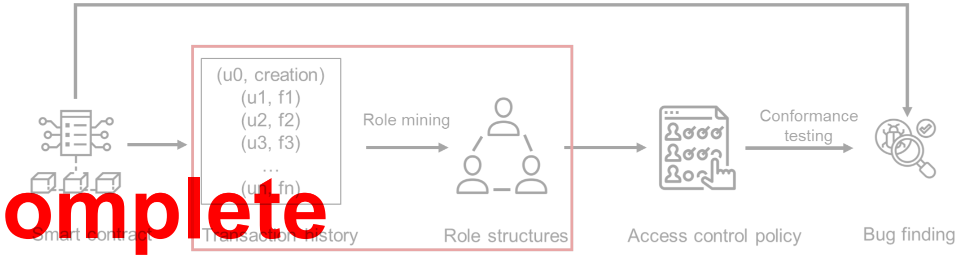



# User access log

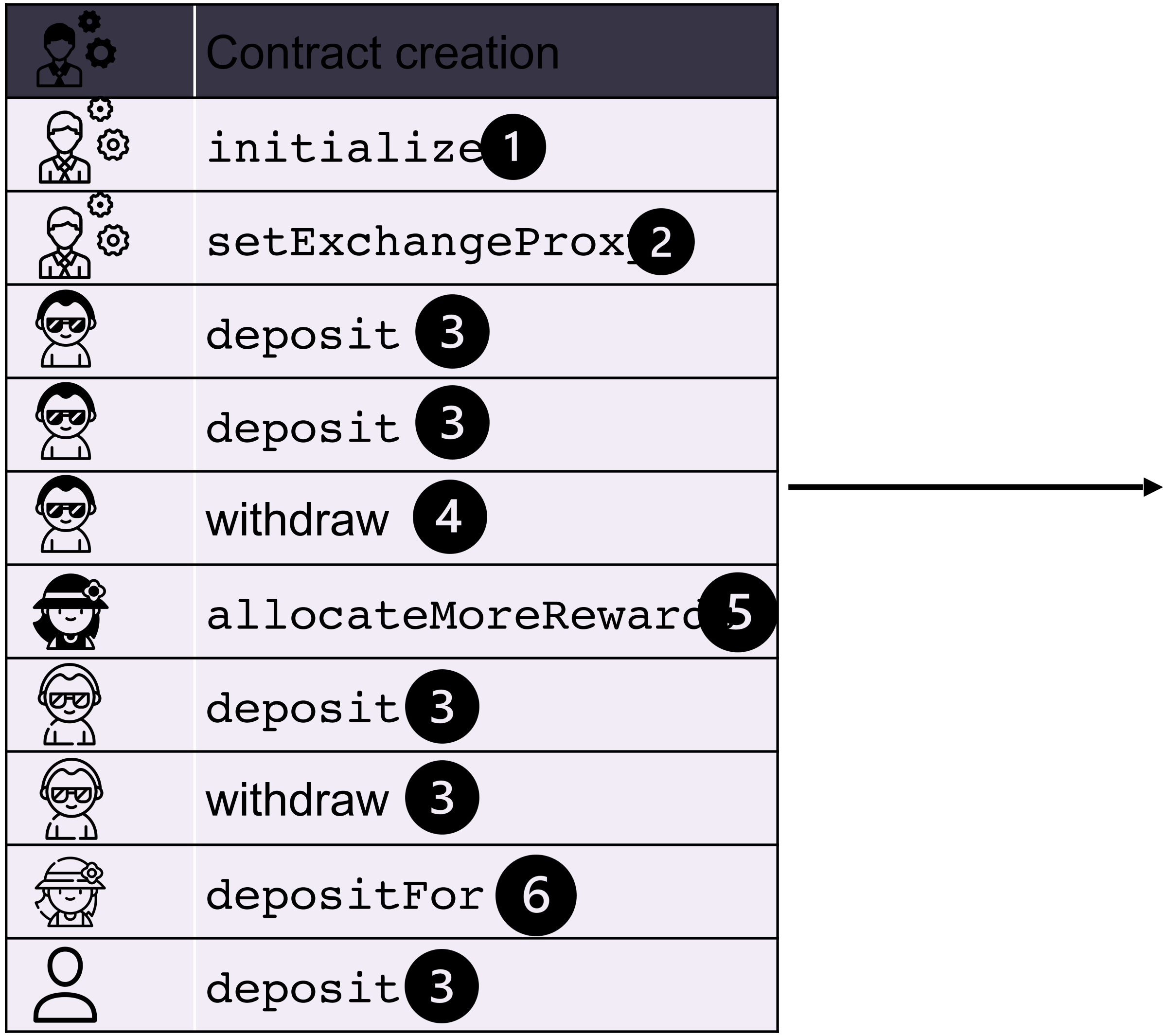
| | Contract creation |
|---|---|
| | initialize 1 |
| | setExchangeProx 2 |
| | deposit 3 |
| | deposit 3 |
| | withdraw 4 |
| | allocateMoreReward 5 |
| | deposit 3 |
| | withdraw 3 |
| | depositFor 6 |
| | deposit 3 |

## User access matrix

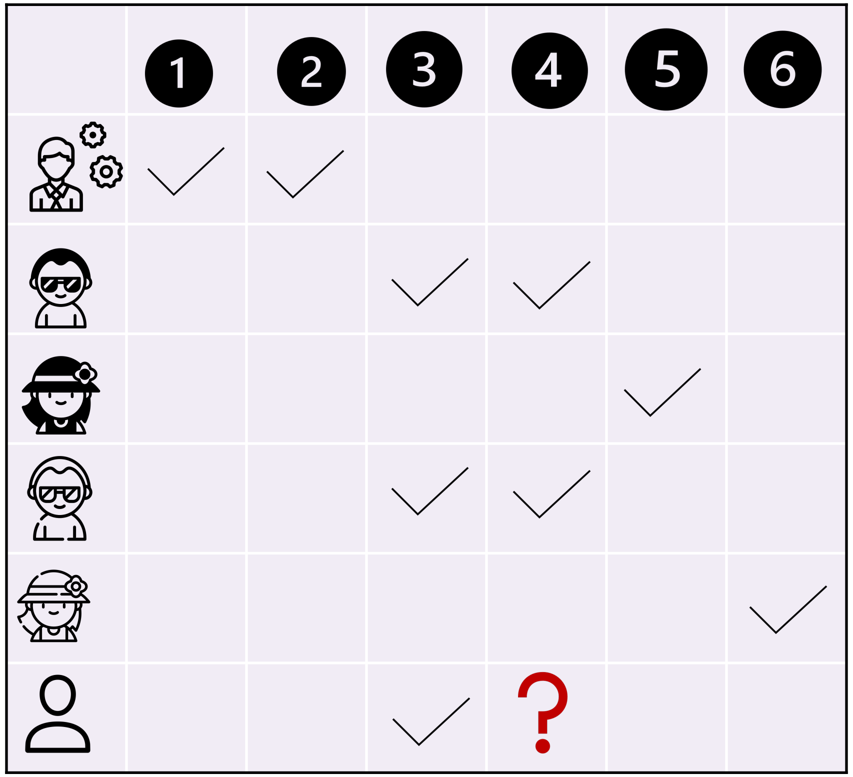
| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| | ✓ | ✓ | | | | |
| | | | ✓ | ✓ | | |
| | | | | | ✓ | |
| | | | ✓ | ✓ | | |
| | | | | | | ✓ |
| | | | ✓ | | | |




# User access log is Incomplete




## User access matrix is Partial



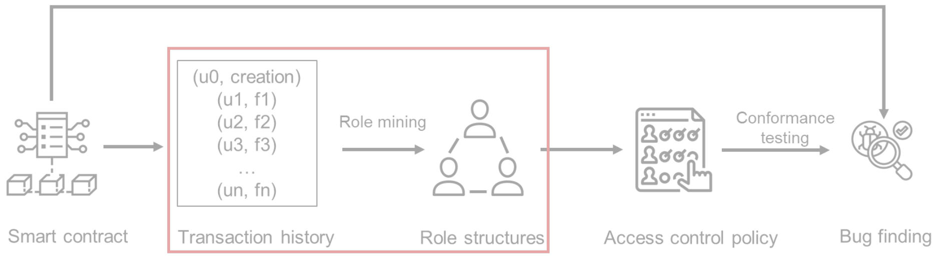
# Role mining from partial observation

- **Assumption**: Users are likely to belong to the same role if they have

  (a) accessed the exact same set of functions.

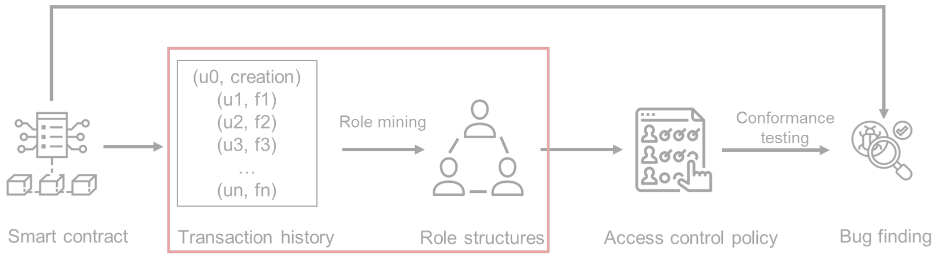
  (b) called common set of functions with similar usage frequency.

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



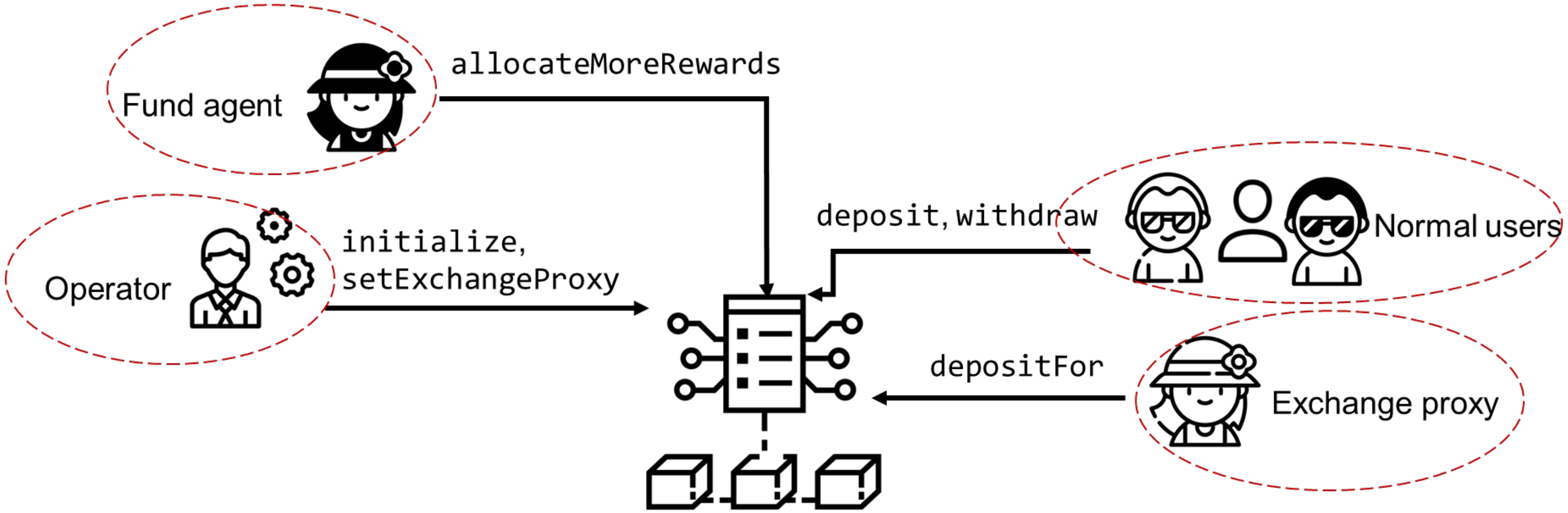
# Role mining from partial observation

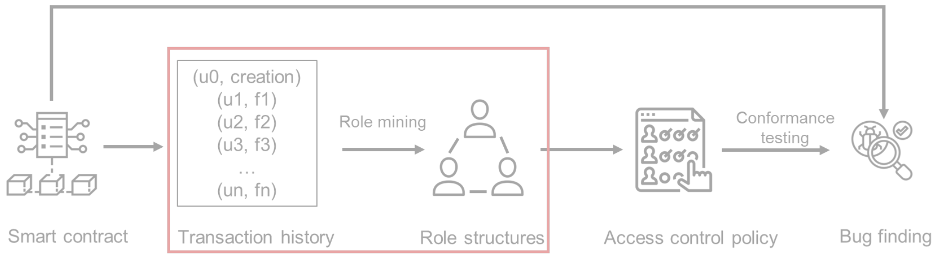
- **Genetic algorithm solution**
  - (a) Frequency similarity metric: measure the chance of a spurious role.
  - (b) Consistency metric: measure the mismatch with the observation.
- **Role mining steps**
  - ➢Infer basic roles: Group users having the same set of function calls
  - ➢Merge basic roles: Combine those with similar frequency patterns



# Role mining result

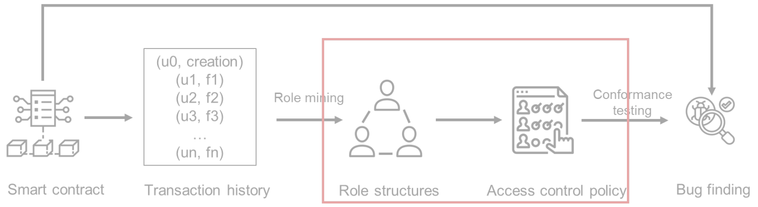
| | Contract creation |
|---|---|
| | initialize |
| | setExchangeProxy |
| | deposit |
| | deposit |
| | withdraw |
| | allocateMoreRewards |
| | deposit |
| | withdraw |
| | depositFor |
| | deposit |



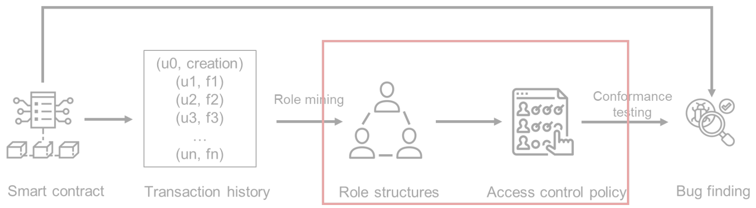
# Role structure

| Users (UA) | Permissions to Functions (PA) |
|---|---|
| { Operator } | { initialize(), setOperator(), setExchangeProxy(), se- tReserveFund(), depositFor(), allocateMoreRewards() } |
| { Exchange proxy } | { setExchangeProxy(), depositFor() } |
| { Fund agent } | { setReserveFund(), allocateMoreRewards() } |
| { Normal Users } | {deposit(), withdraw(), claimRewards() } |

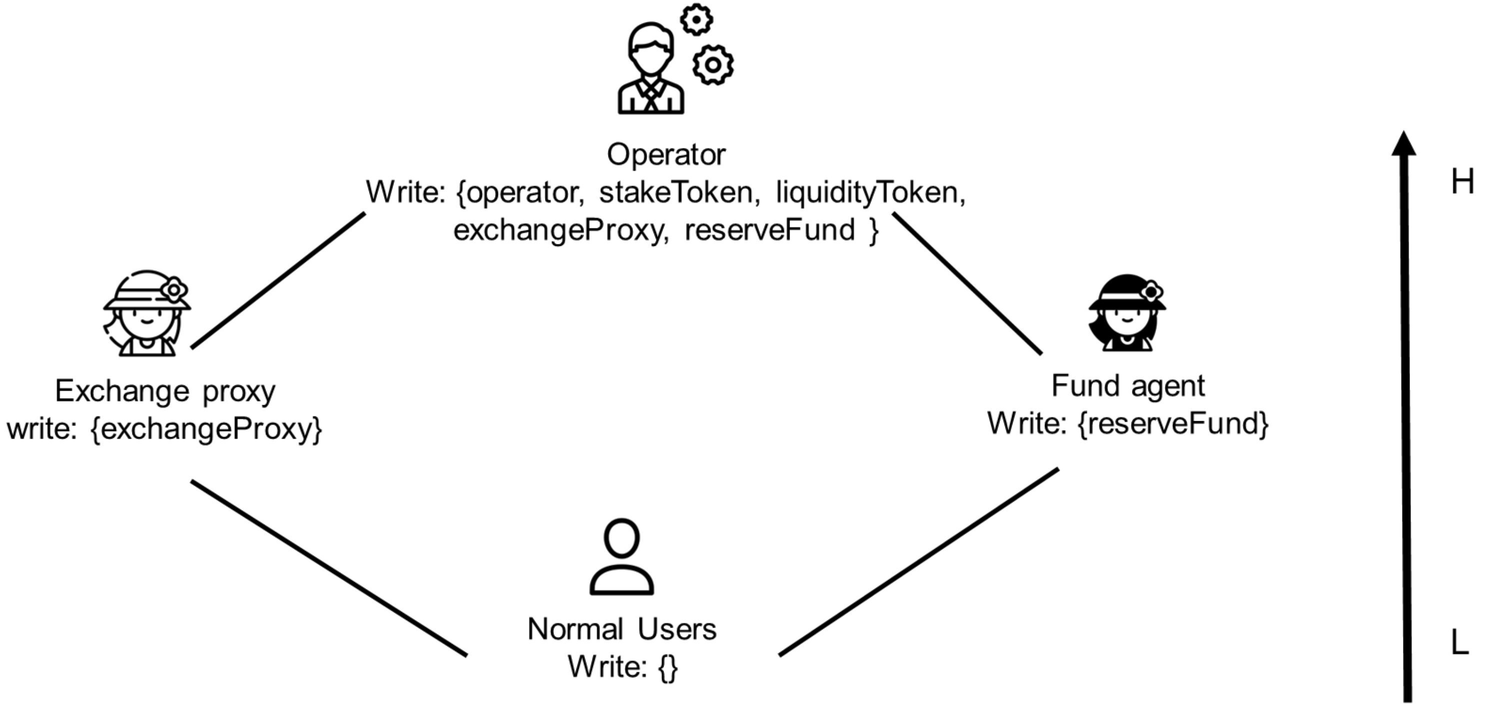


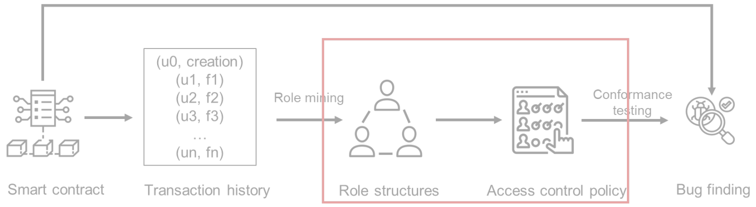
# Role structure

| Users (UA) | Permissions to Functions (PA) | Written State Variables |
| --- | --- | --- |
| { Operator } | { initialize(), setOperator(), setExchangeProxy(), se-tReserveFund(), depositFor(), allocateMoreRewards() } | {operator, stakeToken, liquidityTo-ken, exchangeProxy, reserveFund} |
| { Exchange proxy } | { setExchangeProxy(), depositFor() } | {exchangeProxy} |
| { Fund agent } | { setReserveFund(), allocateMoreRewards() } | {reserveFund} |
| { Normal Users } | {deposit(), withdraw(), claimRewards() } | {} |



# Access control policy (information security lattice)

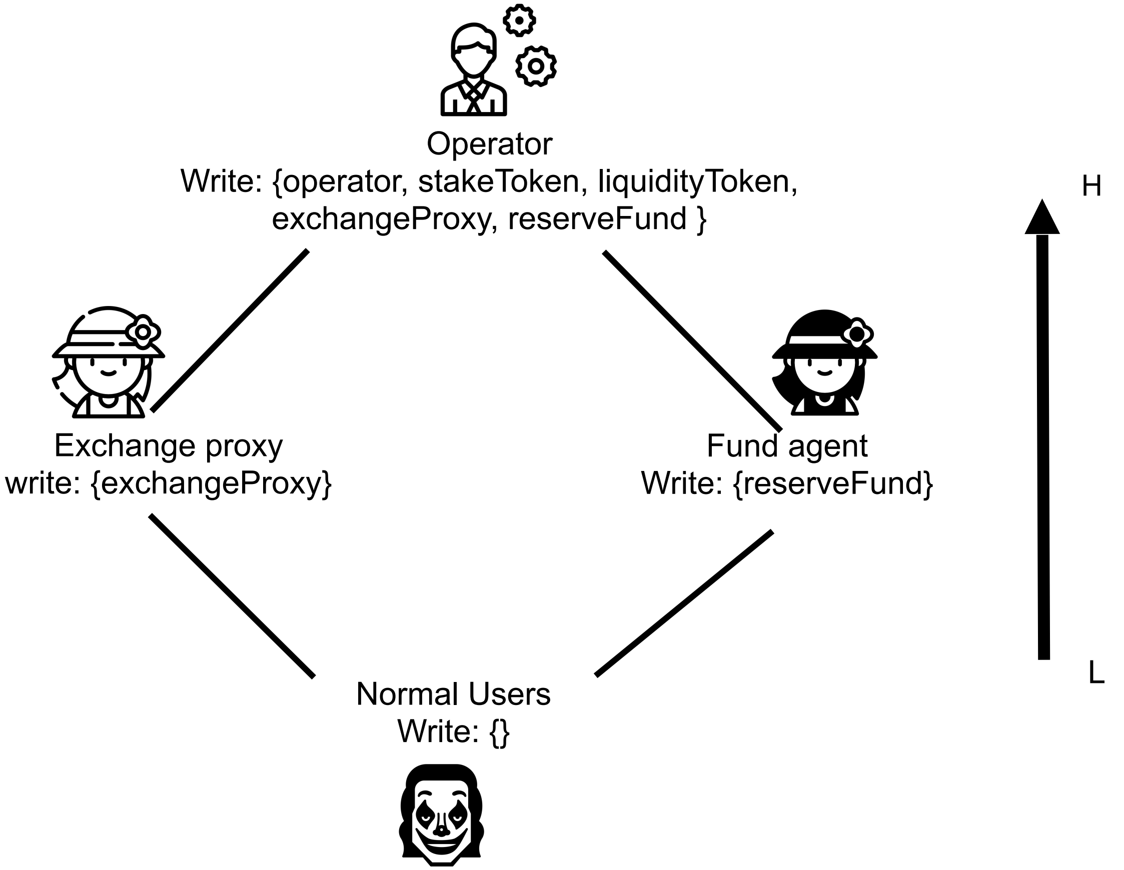


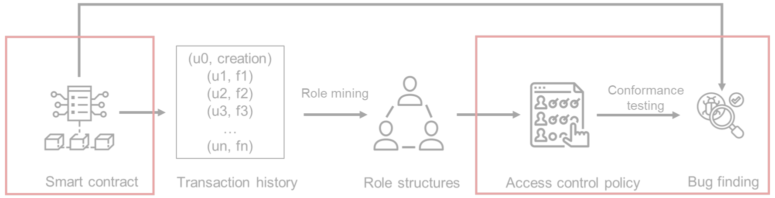



# Conformance testing

- Symbolic execution.
- Concrete value from blockchain snapshot.

| Test Sequences | Policy check |
|---|---|
| ts1 = `setExchangeProxy()` -> `depositFor()` | Safe |
| ts2 = `setReservedFund()` -> `allocateMoreRewards()` | Safe |
| ts3 = `deposit(X)` -> `withdraw(Y)` | Safe |
| ts4 = `initialize()` -> `setExchangeProxy()` | Unsafe |

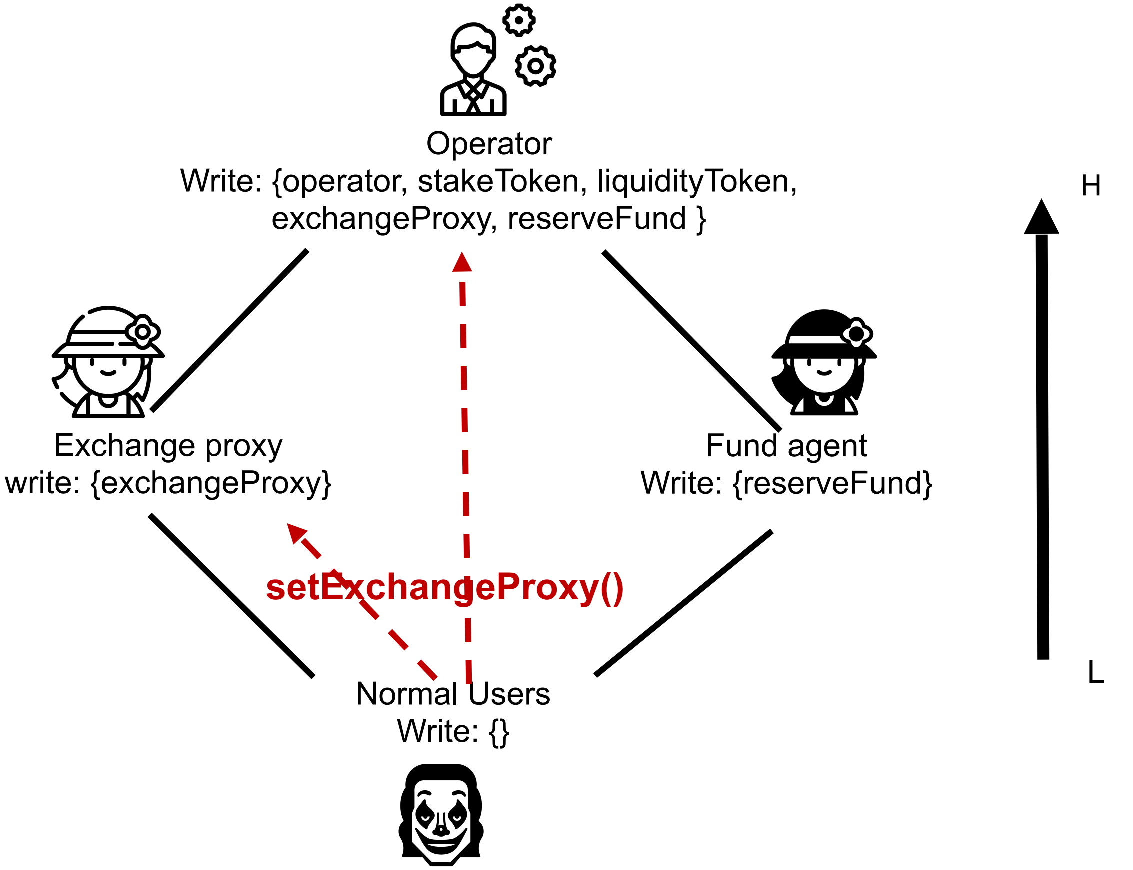


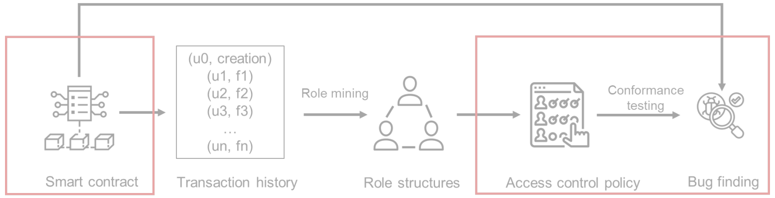



# Conformance testing

- Symbolic execution.
- Concrete value from blockchain snapshot.

| Test Sequences | Policy check |
|---|---|
| **ts1 = `setExchangeProxy(_)` -> `depositFor(_)`** | **Safe** |
| ts2 = `setReservedFund()` -> `allocateMoreRewards()` | Safe |
| ts3 = `deposit(X)` -> `withdraw(Y)` | Safe |
| ts4 = `initialize()` -> `setExchangeProxy()` | Unsafe |

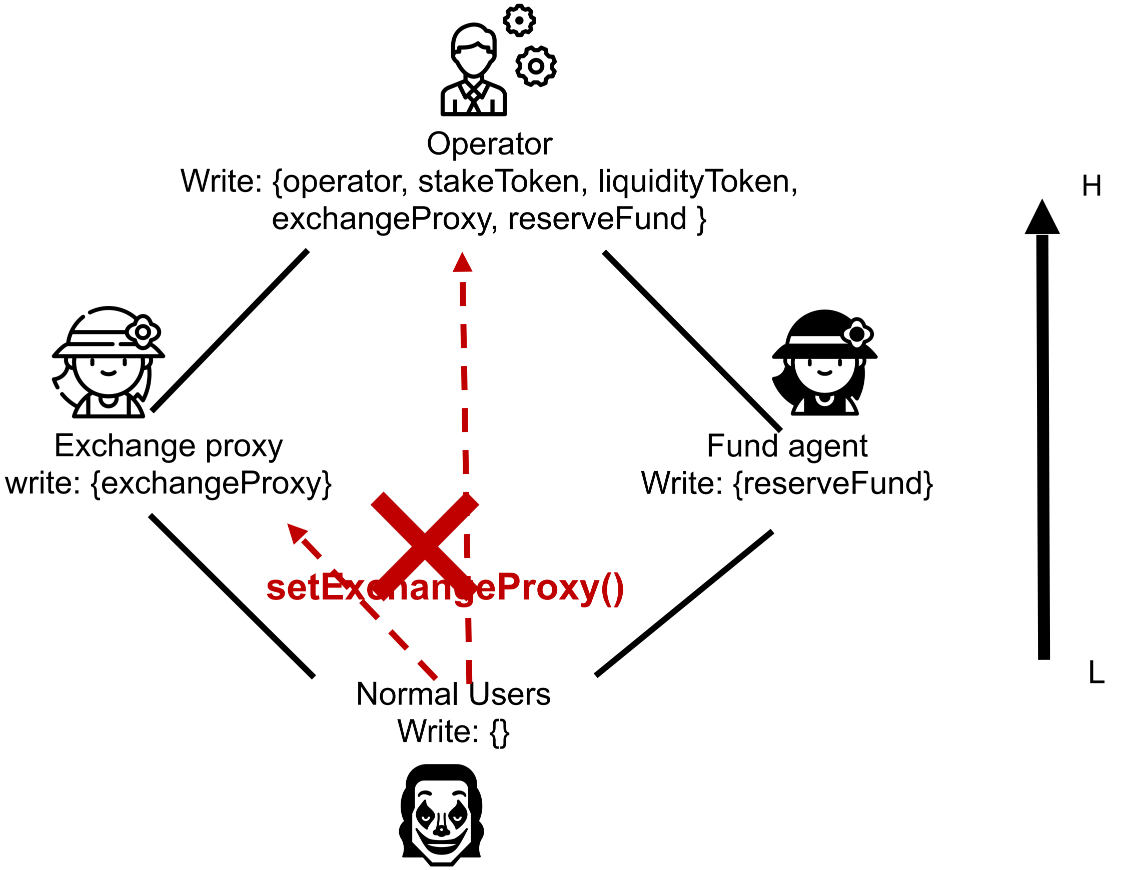


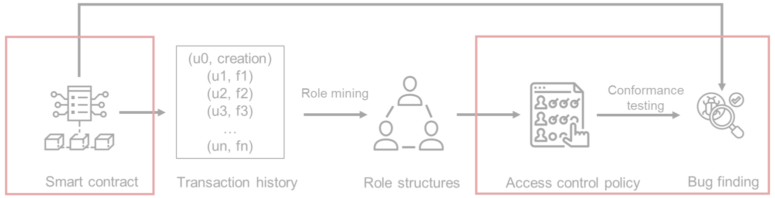



# Conformance testing

- Symbolic execution.
- Concrete value from blockchain snapshot.

| Test Sequences | Policy check |
| --- | --- |
| **ts1 = `setExchangeProxy(_)` -> `depositFor(_)`** | **Safe** |
| ts2 = `setReservedFund()` -> `allocateMoreRewards()` | Safe |
| ts3 = `deposit(X)` -> `withdraw(Y)` | Safe |
| ts4 = `initialize()` -> `setExchangeProxy()` | Unsafe |

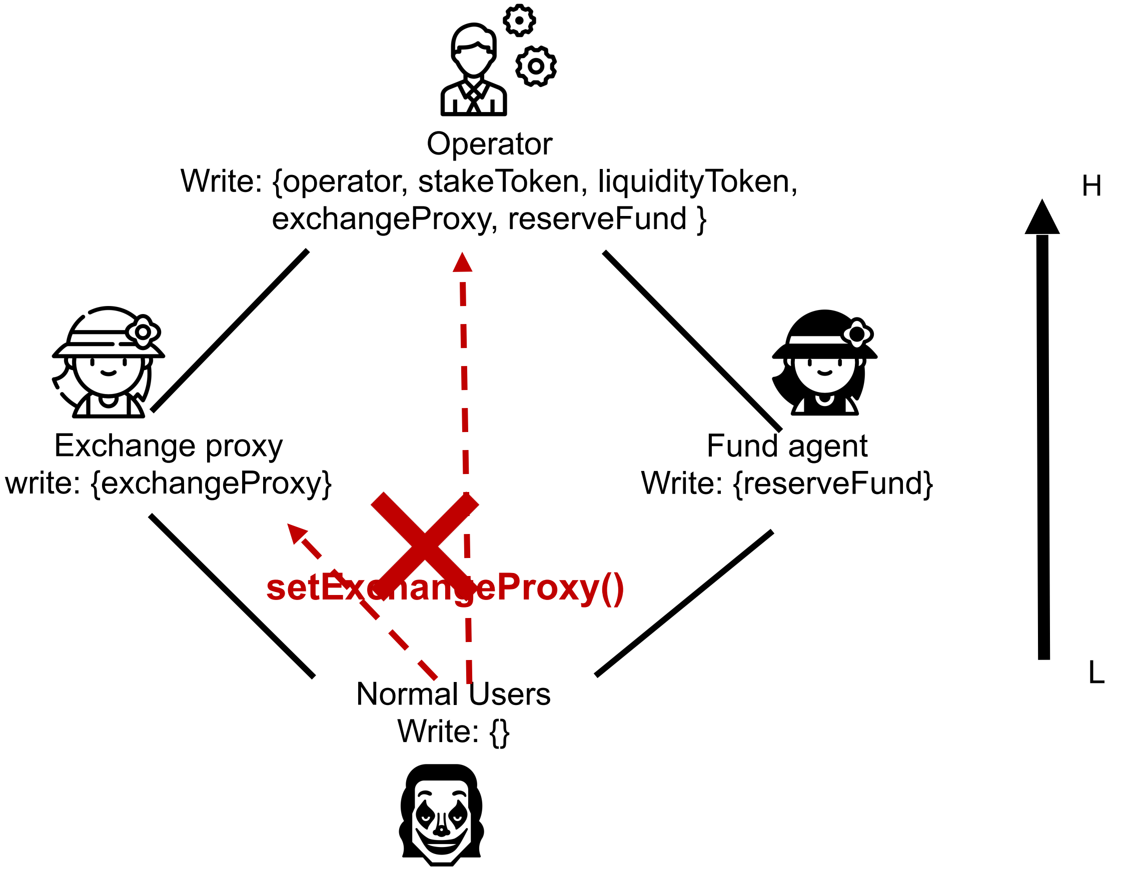


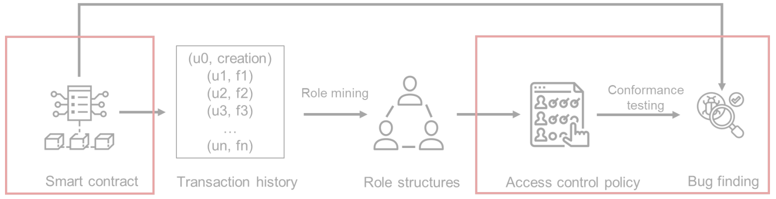



# Conformance testing

- Symbolic execution.
- Concrete value from blockchain snapshot.

| Test Sequences | Policy check |
|---|---|
| ts1 = `setExchangeProxy(_)` -> `depositFor(_)` | **Safe** |
| ts2 = `setReservedFund()` -> `allocateMoreRewards()` | Safe |
| ts3 = `deposit(X)` -> `withdraw(Y)` | Safe |
| ts4 = `initialize()` -> `setExchangeProxy()` | Unsafe |

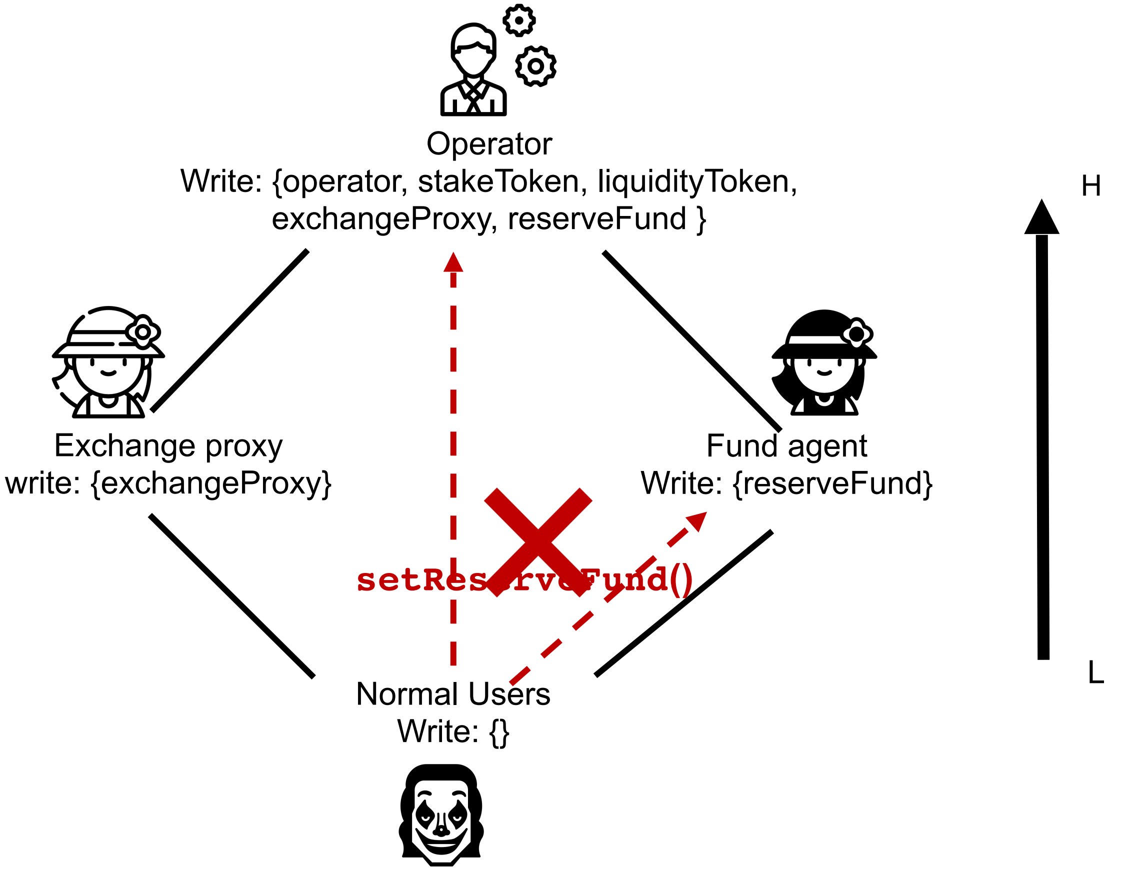


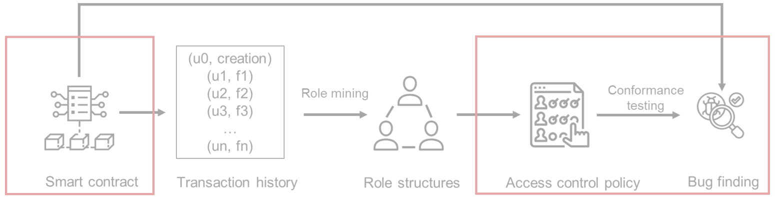



# Conformance testing

- Symbolic execution.
- Concrete value from blockchain snapshot.

| Test Sequences | Policy check |
|---|---|
| ts1 = `setExchangeProxy`(_) -> `depositFor`(_) | Safe |
| **ts2 = `setReservedFund`() -> `allocateMoreRewards`()** | **Safe** |
| ts3 = `deposit`(X) -> `withdraw`(Y) | Safe |
| ts4 = `initialize`() -> `setExchangeProxy`() | Unsafe |

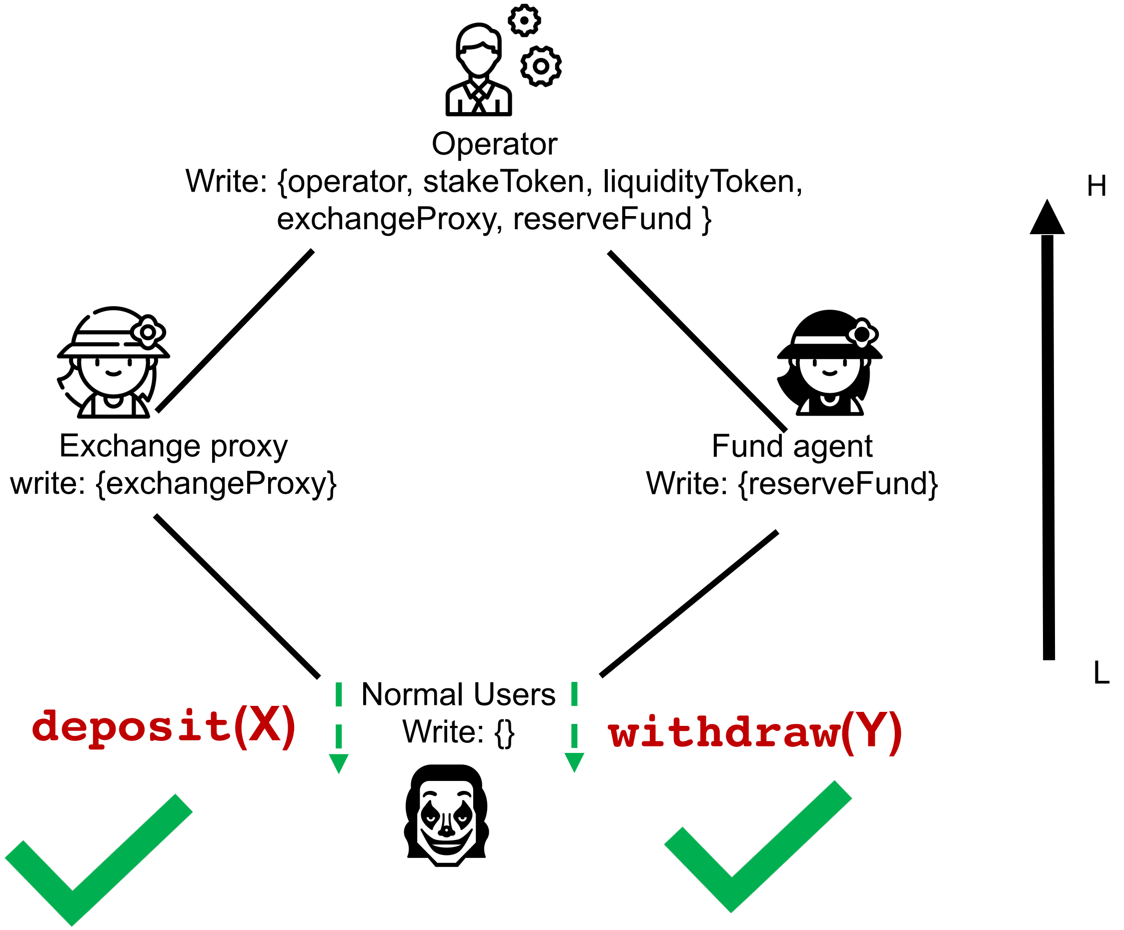


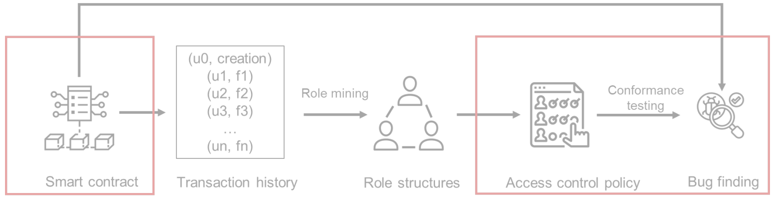



# Conformance testing

- Symbolic execution.
- Concrete value from blockchain snapshot.

| Test Sequences | Policy check |
|---|---|
| ts1 = `setExchangeProxy(_)` -> `depositFor(_)` | Safe |
| ts2 = `setReservedFund()` -> `allocateMoreRewards()` | Safe |
| **ts3 = `deposit(X)` -> `withdraw(Y)`** | **Safe** |
| ts4 = `initialize()` -> `setExchangeProxy()` | Unsafe |

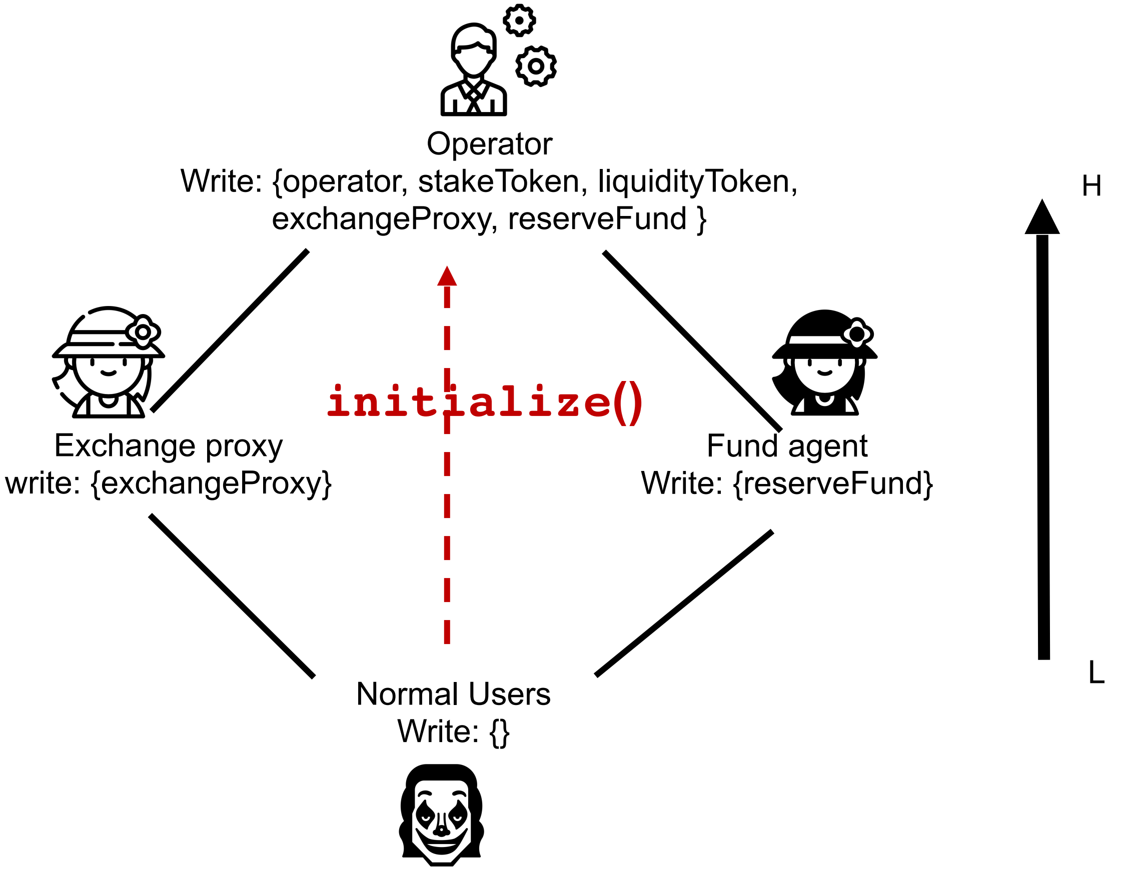


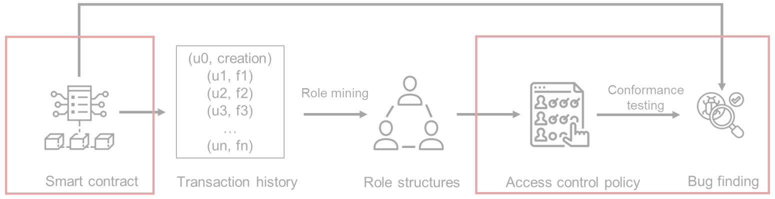



# Conformance testing

- Symbolic execution.
- Concrete value from blockchain snapshot.

| Test Sequences | Policy check |
| --- | --- |
| ts1 = `setExchangeProxy`(_) -> `depositFor`(_) | Safe |
| ts2 = `setReservedFund`() -> `allocateMoreRewards`() | Safe |
| ts3 = `deposit`(X) -> `withdraw`(Y) | Safe |
| **ts4 = `initialize()` -> `setExchangeProxy()`** | **Unsafe** |

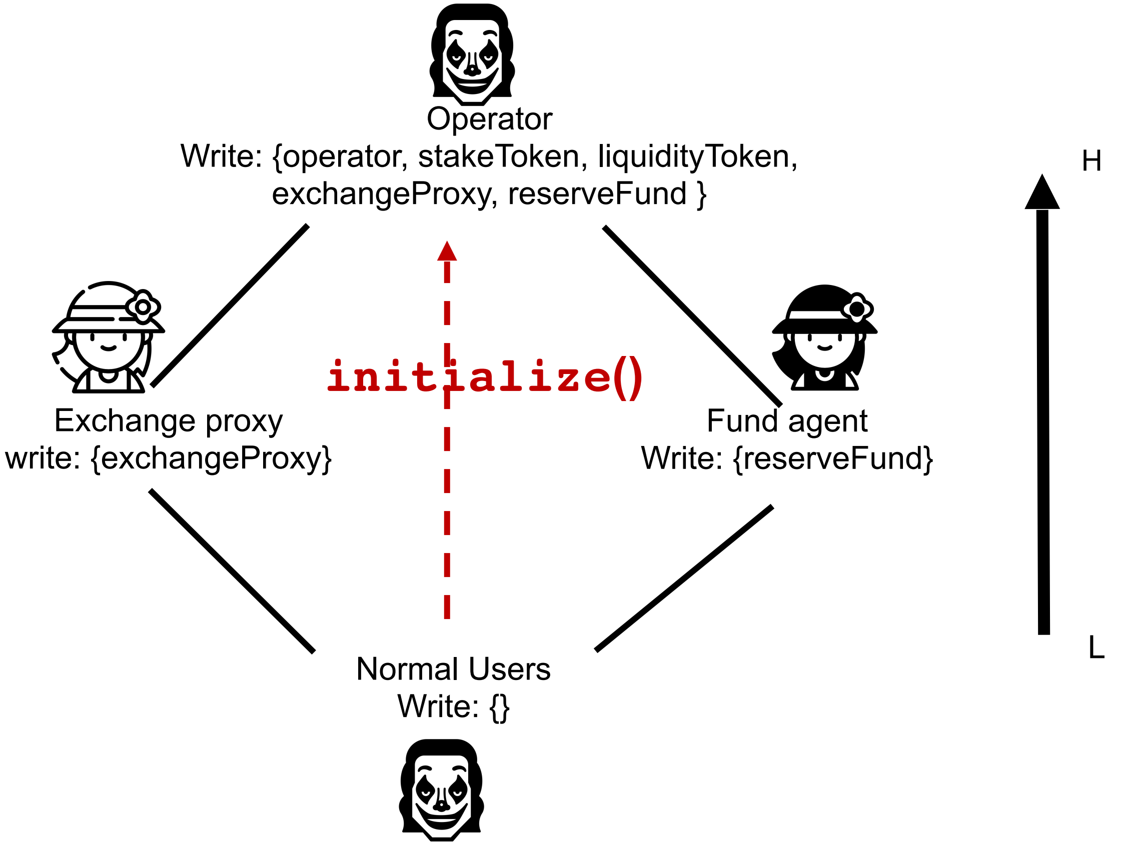


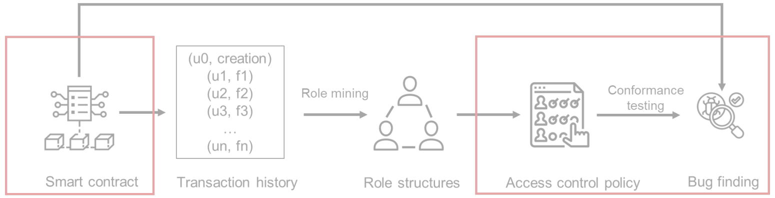



# Conformance testing

- Symbolic execution.
- Concrete value from blockchain snapshot.

| Test Sequences | Policy check |
| --- | --- |
| ts1 = `setExchangeProxy(_)` -> `depositFor(_)` | Safe |
| ts2 = `setReservedFund()` -> `allocateMoreRewards()` | Safe |
| ts3 = `deposit(X)` -> `withdraw(Y)` | Safe |
| **ts4 = `initialize()` -> `setExchangeProxy()`** | **Unsafe** |

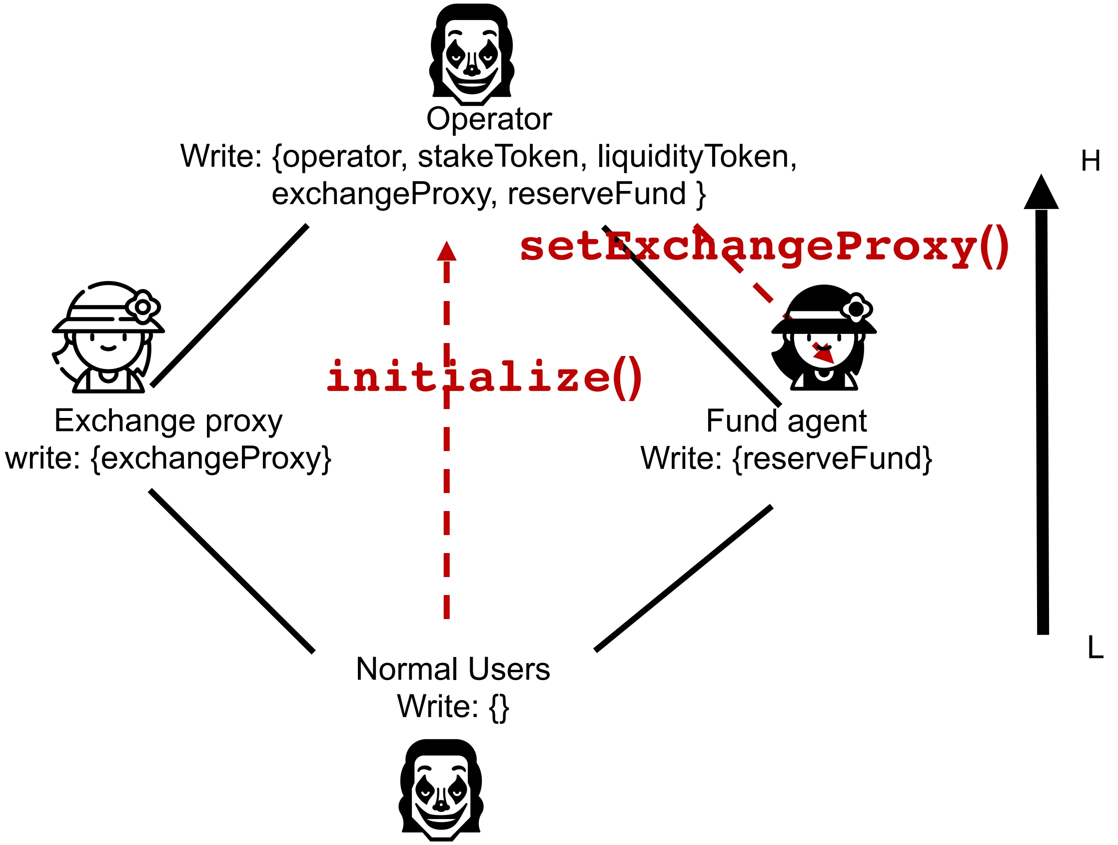


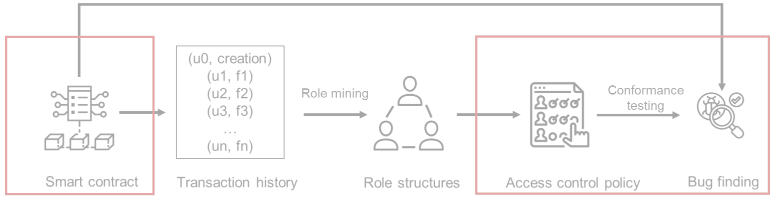



# Conformance testing

- Symbolic execution.
- Concrete value from blockchain snapshot.

| Test Sequences | Policy check |
|---|---|
| ts1 = `setExchangeProxy`(_) -> `depositFor`(_) | Safe |
| ts2 = `setReservedFund`() -> `allocateMoreRewards`() | Safe |
| ts3 = `deposit`(X) -> `withdraw`(Y) | Safe |
| **ts4 = `initialize()` -> `setExchangeProxy()`** | **Unsafe** |



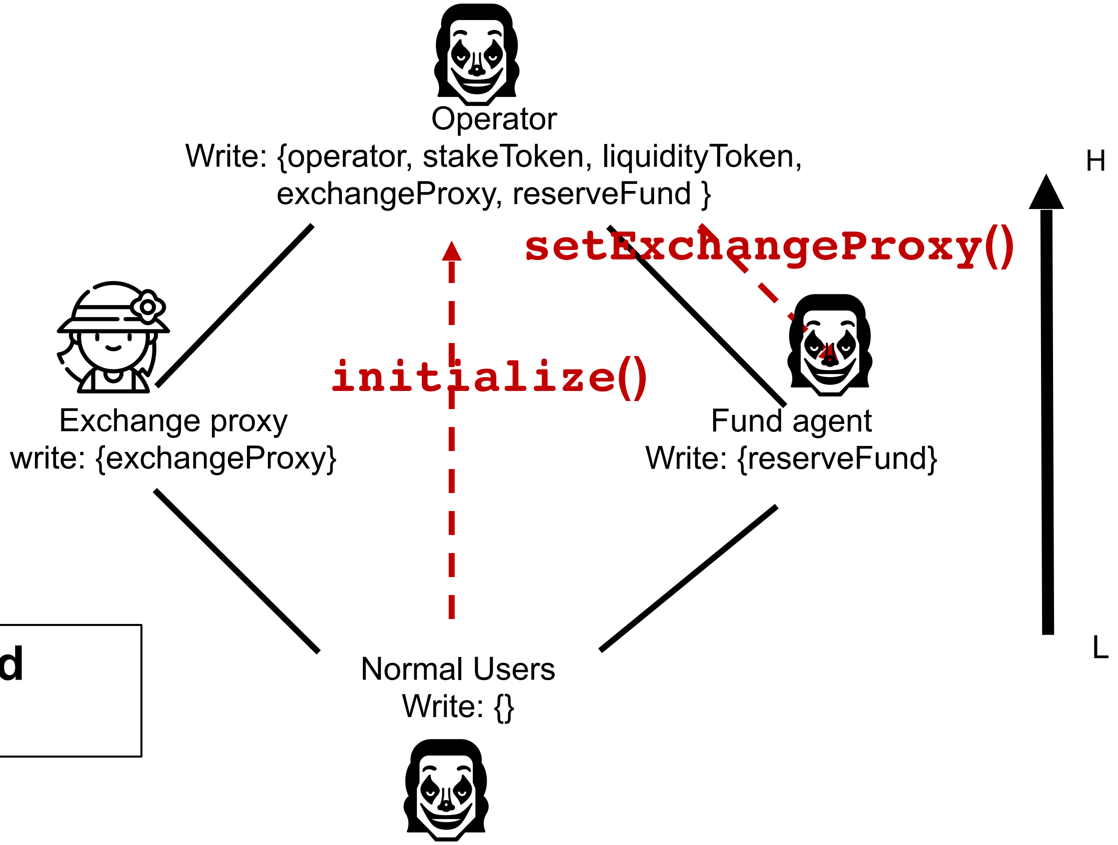



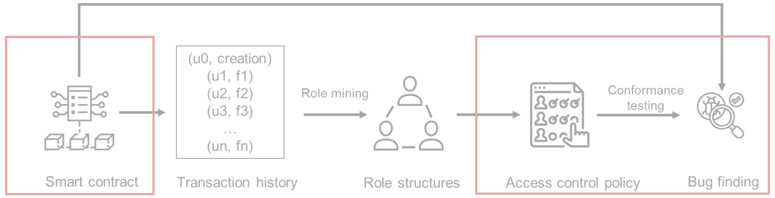
# Conformance testing

- Symbolic execution.
- Concrete value from blockchain snapshot.

| Test Sequences | Policy check |
|---|---|
| ts1 = `setExchangeProxy(_)` -> `depositFor(_)` | Safe |
| ts2 = `setReservedFund()` -> `allocateMoreRewards()` | Safe |
| ts3 = `deposit(X)` -> `withdraw(Y)` | Safe |
| **ts4 = `initialize()` -> `setExchangeProxy()`** | **Unsafe** |

**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

| Approach | Run time (s) | Number of roles | Number of mined roles per roles in ground truth |
|---|---|---|---|
| HPr | 0.21 | 8.28 | 2.69 |
| ORCA | 5.08 | 21.96 | 7.17 |
| HM | 49.54 | 19.04 | 6.37 |
| GO | 191.72 | 15.34 | 4.86 |
| **SPCon (0.4, 0.6)** | 31.69 | **7.00** | **2.27** |
| **SPCon (0.5, 0.5)** | 33.10 | **4.64** | **1.54** |
| **SPCon (0.6, 0.4)** | 34.55 | **3.52** | **1.11** |

**SPCon can accurately and efficiently reverse engineer likely user roles of smart contracts**

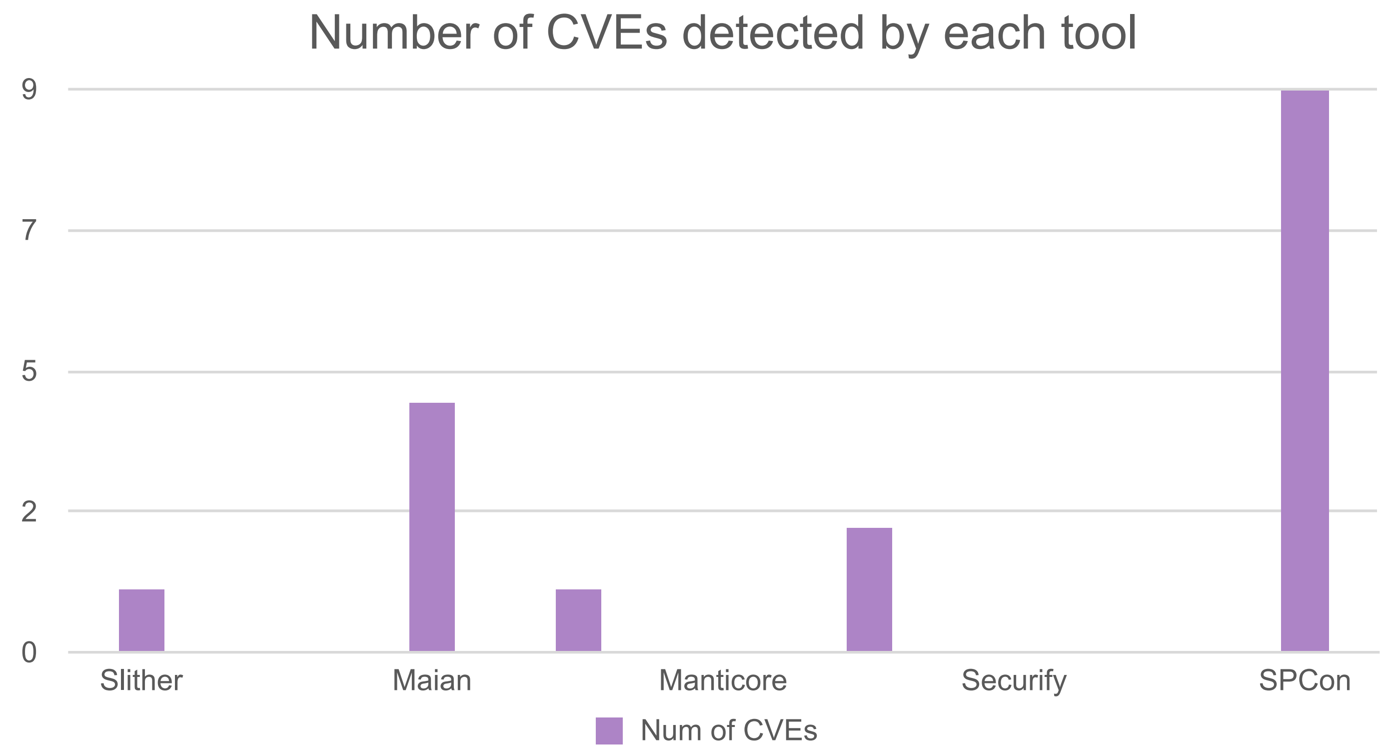
# RQ2: Performance in permission bug finding on the benchmark SmartBugs.

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

**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.