SmartState : Detecting State-reverting Vulnerabilities in Smart Contracts via Fine-grained State-dependency Analysis

Zeqin Liao, Sicheng Hao, Yuhong Nan*, Zibin Zheng

Friday, February 23,
2024

中山大學
SUN YAT-SEN UNIVERSITY

LAB
WWW.INPLUSLAB.COM

Smart Contract

In 2022, over 7.75 million smart contracts were deployed on Ethereum 

- Running on blockchain
- Components of DApps
- Finance(DeFi), Game(GameFi) Token, NFT, Exchanges,…

- The DAO: $60M theft
- Poly Network: $611M lost
- Akutars: $34M locked

State & Transaction

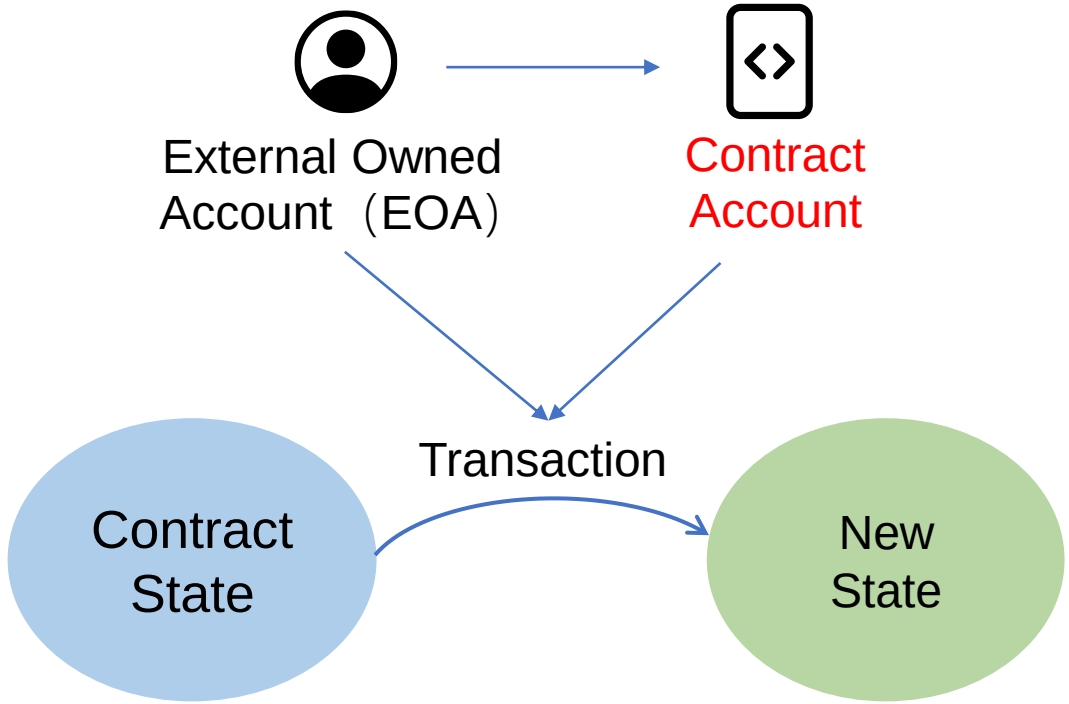
States are modified by transactions

Contract state

```
contract TokenGame {
  mapping(address => uint256) public SheepToken;
  mapping(address => uint256) public WolfToken;
  mapping(address => uint256) public Earning;
  function Withdraw(address account,unit amount) public{
    require(SheepToken[account]>0||WolfToken[account]>0);
    tranferForm(address(this), account, amount); }

  function PlaytoEarn(address account,unit tokenId)
      public{
    if(isWolf(tokenTraits[tokenId]))
      Earning[account]=Earning[account]*(2-Rate); }
  ...
}
```

Entry point



State-reverting Mechanism

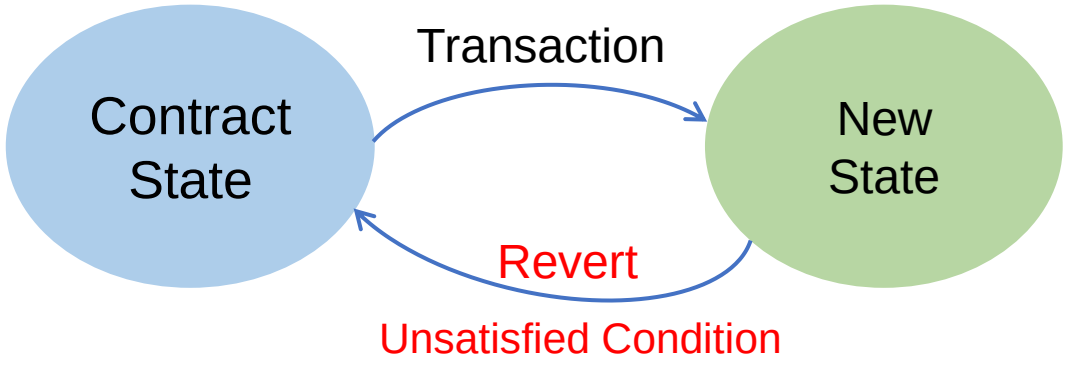
Revert states under unsatisfied condition

```
function withdraw() public {
    require(tx.origin == msg.sender)
    payable(msg.sender).transfer(balance[msg.sender]);
    }

function buy(uint amount) public payable {
    if (amount > msg.value / 2 ether)
        revert("Not enough Ether provided.");

    require(
        amount <= msg.value / 2 ether,
        "Not enough Ether provided."
    );
    // Perform the purchase.
}
```

Only EOA can invoke this function

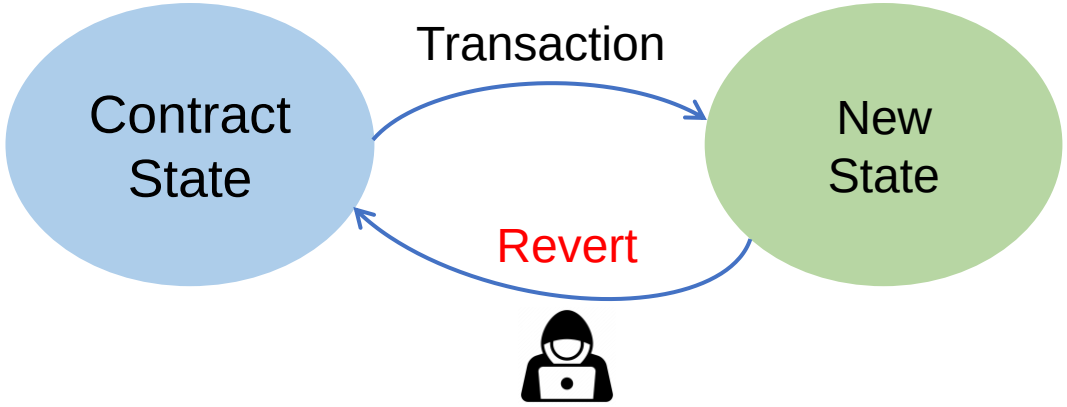


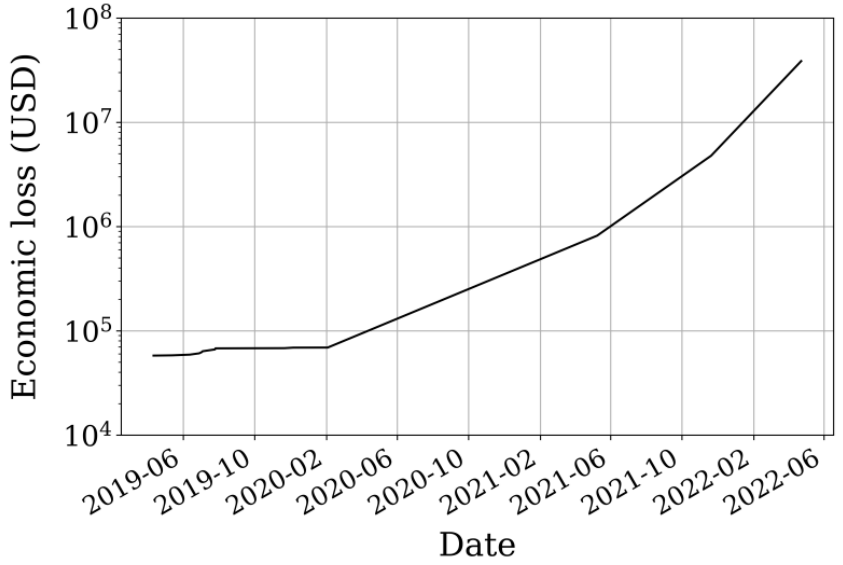
State-reverting Vulnerability

Utilize State-reverting Mechanism to attack

If the result of the transaction is not as expected

Rollback it!

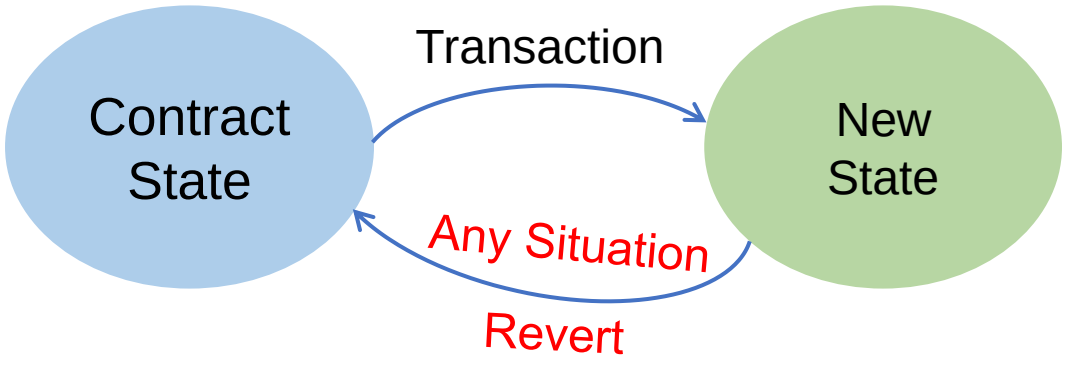




Economic loss caused by SRVs.

SRV: DoS

- **Rollback the whole transaction**




```
contract Refunder {

address[] private refundAddresses;
mapping (address => uint) public refunds;

  function refundAll() public {
    for(uint x; x < refundAddresses.length; x++) {
      uint fund = refunds[refundAddresses[x]]
      require(refundAddresses[x].send(fund));
    }}
}
```

Call fallback function in attacker contract

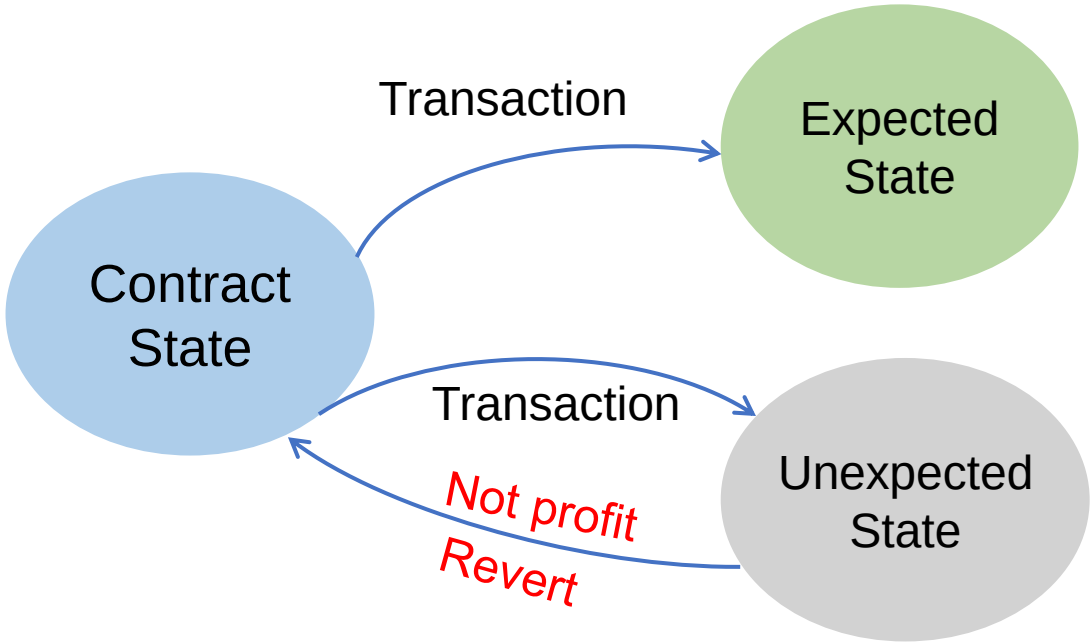
```
contract Attacker {

  function() public {
    revert();
    }
}
```

The whole transaction fails

SRV: Profit-gain

Rollback unexpected results




```
contract TokenGame {
  mapping(address => uint256) public SheepToken;
  mapping(address => uint256) public WolfToken;
  function MintToken(address account) public {
    uint256 seed= (random () >> 245) % 10;
    //A random number determines gambling results
    if ( seed != 0) {
      SheepToken[account]++;}
    else{
      //The state variable manipulated by attacker
      WolfToken[account]++;} }
```

Random drawing

Only want this !

```
contract Attacker {
  function onlyWolf(TokenGame target, tokenId) public{
    unit256 Before = WolfToken.balanceOf(address(this));
    target.MintToken(tokenId);
    unit256 After = WolfToken.balanceOf(address(this));
    // Reverting the unexpected gambling result
    require(After > Before); }
}
```

Require profit-gain result

SRV: Profit-gain

State Dependencies

```
contract TokenGame {
  mapping(address => uint256) public SheepToken;
  mapping(address => uint256) public WolfToken;
  mapping(address => uint256) public Earning;
  ...
  function MintToken(address account) public {
    uint256 seed= (random () >> 245) % 10;
    if ( seed != 0) {
      SheepToken[account]++;}
    else{
      WolfToken[account]++;} }

  function Withdraw(address account,unit amount) public{
    require(SheepToken[account]>0||WolfToken[account]>0);
    tranferForm(address(this), account, amount); }

  function PlaytoEarn(address account,unit tokenId)
       public{
    if(isWolf(tokenTraits[tokenId]))
      Earning[account]=Earning[account]*(2-Rate); }
  ...
}
```

TSD

ASD

State R&W relationship

Control-flow relationship

Not enough

Assertion-related state dependency (ASD)

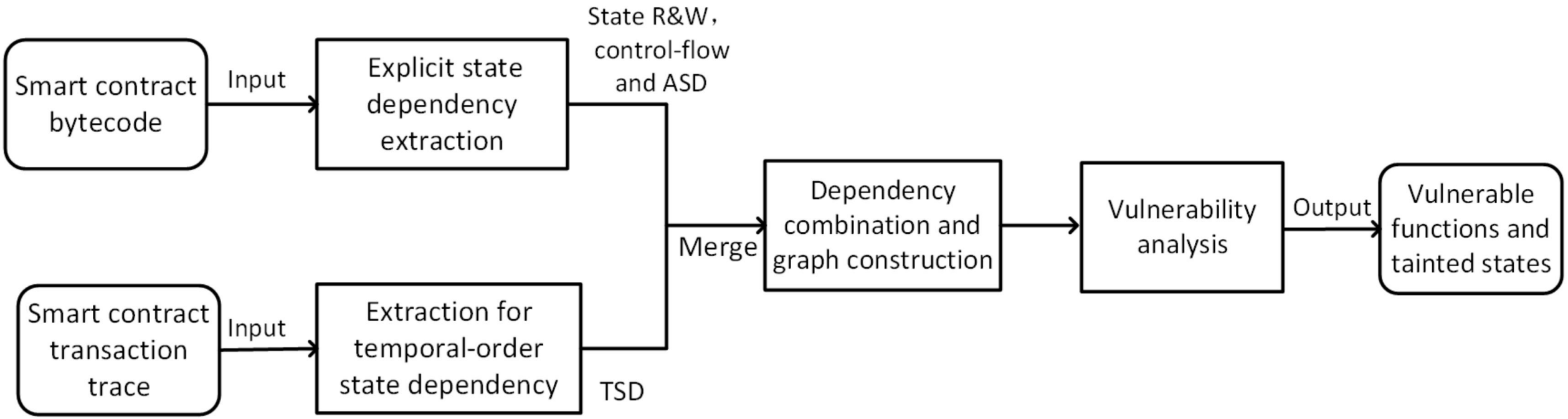
Temporal-ordered state dependency (TSD)

Merge

Fine-grained State-dependency Graph

SmartState

Overview



SmartState

Assertion-related state dependency (ASD)

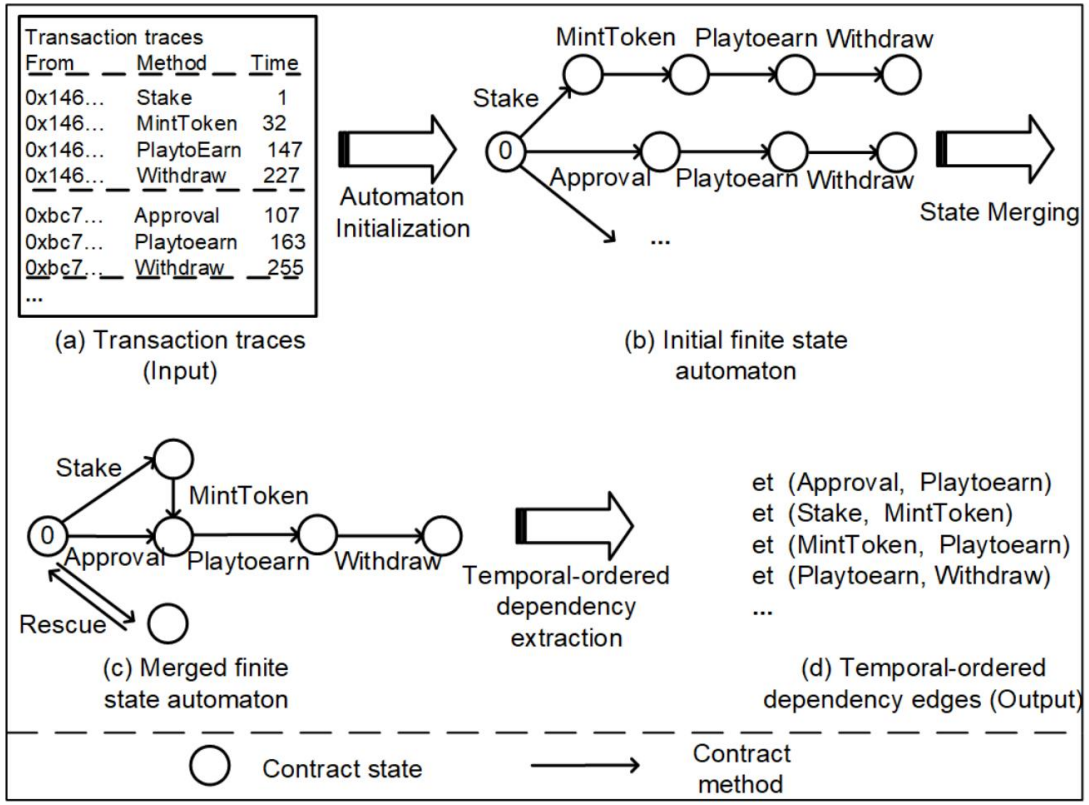
- A function M_r reads the state variable S_d as a condition within the assertion statement (i.e., revert, assert, require)
- Another function Mw writing on the same state variable S_d
- $Er = \{er\ (M_r, M_w) | M_r, M_w \in M\}$.

```
6  function MintToken(address account) public {
7    uint256 seed= (random () >> 245) % 10;
8    if ( seed != 0) {
9      SheepToken[account]++;}
10   else{
11     WolfToken[account]++;} }
12
13 function Withdraw(address account,uint amount) public{
14   require(SheepToken[account]>0||WolfToken[account]>0);
15   tranferFrom(address(this), account, amount); }
```

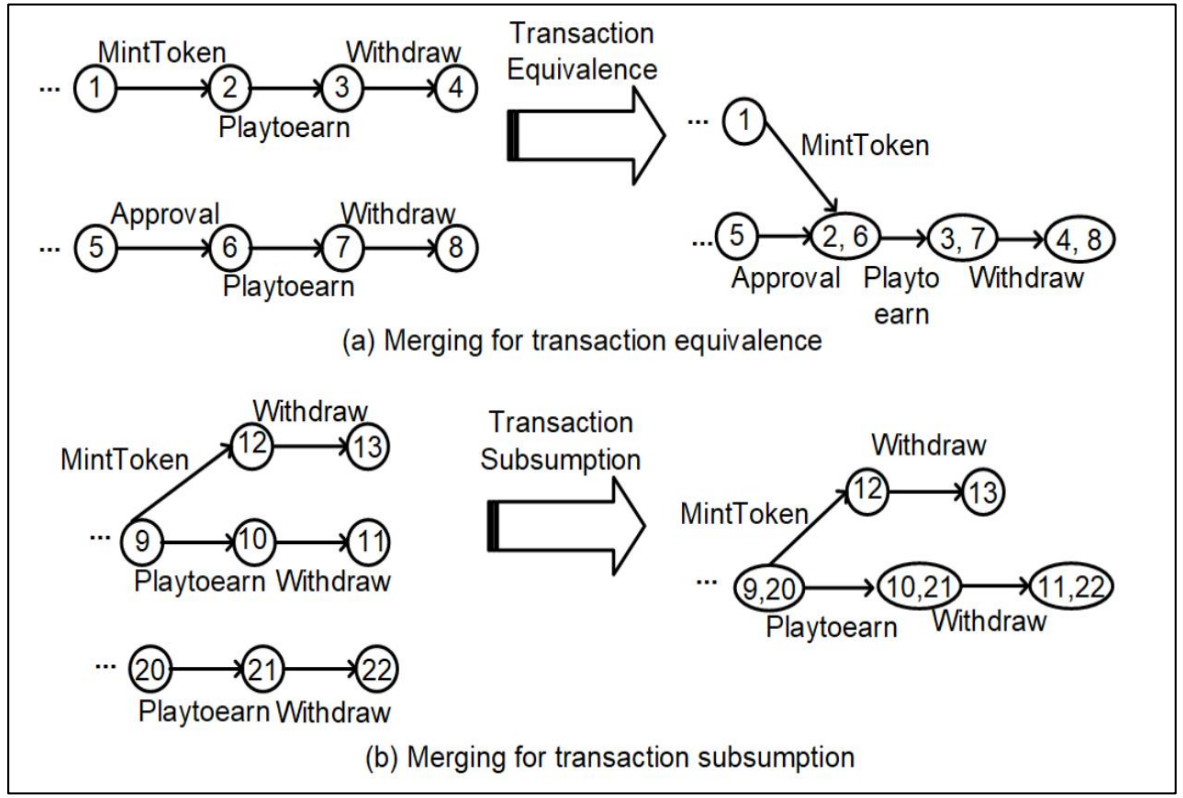
ASD

SmartState

Temporal-order state dependency (TSD)



The finite state machine

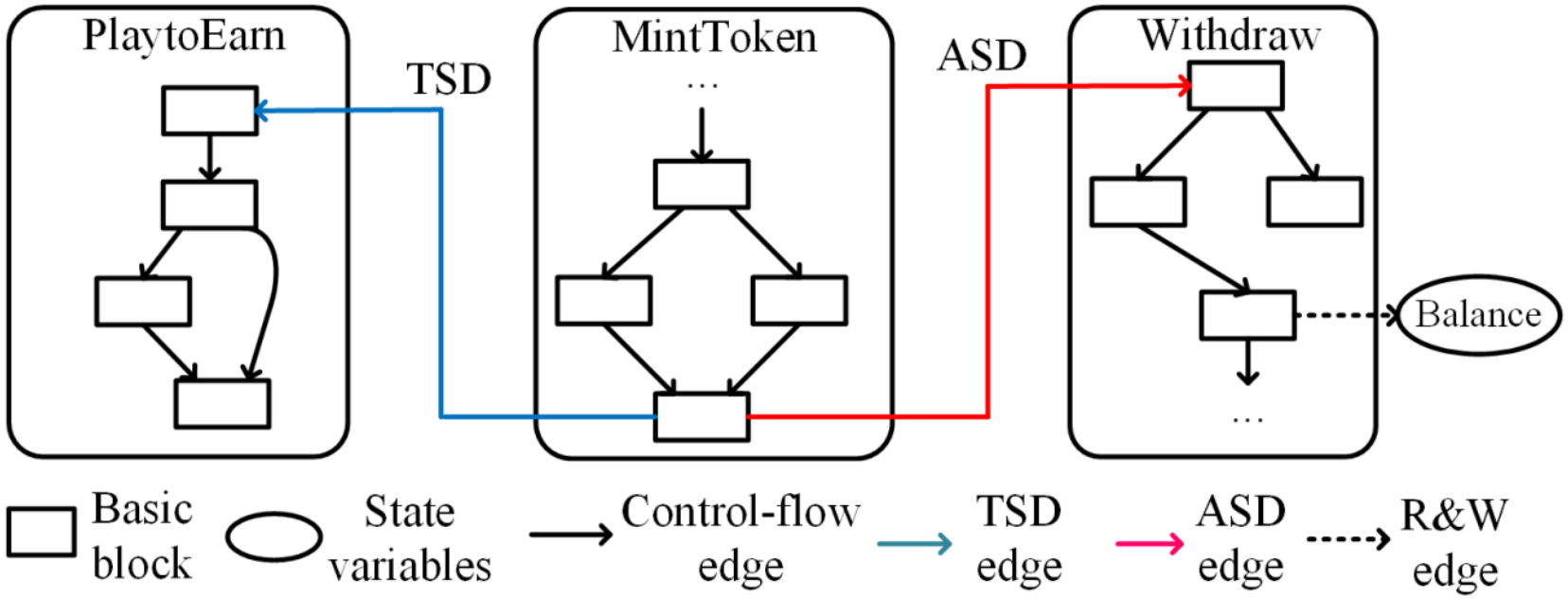


illustrating transaction equivalence and transaction subsumption

SmartState

Fine-grained SDG construction

- Control flow + State R&W dependency + ASD + TSD



SmartState

Vulnerability Detection

```
1 contract TokenGame {
2   mapping(address => uint256) public SheepToken;
3   mapping(address => uint256) public WolfToken;
4   mapping(address => uint256) public Earning;
5   ...
6   function MintToken(address account) public {
7     uint256 seed= (random () >> 245) % 10;
8     if ( seed != 0) {
9       SheepToken[account]++;}
10    else{
11      WolfToken[account]++;} }
12
13  function Withdraw(address account,unit amount) public{
14    require(SheepToken[account]>0||WolfToken[account]>0);
15    tranferForm(address(this), account, amount); }
16
17  function PlaytoEarn(address account,unit tokenId)
        public{
18    if(isWolf(tokenTraits[tokenId]))
19      Earning[account]=Earning[account]*(2-Rate); }
20  ...
21 }
```

Entry point

SRV indicator

Tainted variable

Tainted variable

Vulnerability indicator rules

Vulnerability type	Function selection rule
R1-Profit-gain attack	$isRandomness(var_{state}) \vee isLackof(C_{acc})$
R2-DoS attack	$(isinLoop(externalcall) \vee isModified(var_{state})) \vee isLackof(C_{acc})$

Taint sources and sinks

	Type	EVM instructions
Source	(1) Parameter passed by contract invoker	CALLDATALOAD, CALLDATACOPY, CALLER, ORIGIN, CALLVALUE, CALLDATASIZE
	(2) Parameter of public function	Public, External
Sink	(1) External calls	CALL, CALLCODE, STATICCALL, DELEGATECALL
	(2) State variables	CALL, CALLCODE, STATICCALL, DELEGATECALL

Traces: (1) MintToken → {$SheepToken$,$WolfToken$}, (2) MintToken → Playtoearn → {$Earning$}, (3) MintToken → Withdraw → {$Balance$}.

Evaluation

Effectiveness of SmartState

- SmartState achieves good precision and recall

Attack exploits SRV	Precision			Recall		
	TP	FP	rate	TP	FN	rate
Profit-gain attack	24	5	82.76%	24	4	85.71%
DoS attack	58	7	89.23%	58	6	90.63%
Total	82	12	87.23%	82	10	89.13%

Evaluation

Effectiveness of ASD and TSD

- ASD and TSD are useful for detecting SRVs

Approach	SmartState w/o ASD and TSD			SmartState w/o TSD			SmartState		
	TP	FN	recall	TP	FN	recall	TP	FN	recall
Profit-gain attack	12	16	42.86%	18	10	64.28%	24	4	85.71%
Dos attack	42	22	65.63%	53	11	82.81%	58	6	90.63%
Total	54	38	58.70%	71	21	77.17%	82	10	89.13%

Summary

Detect SRVs effectively

- SmartState proposed a fine-grained state dependency graph with assertion-related dependency and transaction-order dependency
- SmartState achieved 87.23% on precision and 89.13% on recall
- SmartState identified 406 new SRVs in the real world

https://github.com/InPlusLab/SmartState

THANKS

SmartState : Detecting State-reverting Vulnerabilities in Smart Contracts via Fine-grained State-dependency Analysis

Evaluation

Large-scale Analysis

- Reports 771 warnings, including 651 TPs and 120.
- 406 are new SRVs
- 11 SRVs exist in the popular smart contracts
- Affects a total asset of 428,600 USD

SRV: Profit-gain

Rollback unexpected results

```
contract TokenGame {
  mapping(address => uint256) public SheepToken;
  mapping(address => uint256) public WolfToken;
  mapping(address => uint256) public Earning;
  ...
  function MintToken(address account) public {
    uint256 seed= (random () >> 245) % 10;
    if ( seed != 0) {
      SheepToken[account]++;}
    else{
      WolfToken[account]++;} }

  function Withdraw(address account,unit amount) public{
    require(SheepToken[account]>0||WolfToken[account]>0);
    tranferForm(address(this), account, amount); }

  function PlaytoEarn(address account,unit tokenId)
       public{
    if(isWolf(tokenTraits[tokenId]))
      Earning[account]=Earning[account]*(2-Rate); }
  ...
}
```

More valuable

Random drawing

Only want this !

Gain profit