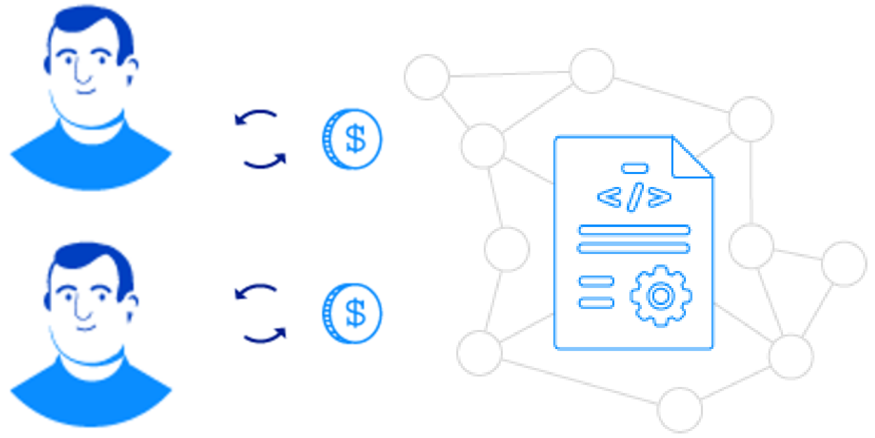
SmartCoCo: Checking Comment-code Inconsistency in Smart Contracts via Constraint Propagation and Binding

Sicheng Hao, Yuhong Nan, Zibin Zheng, Xiaohui Liu

Tuesday, September 12, 2023

Smart Contract



Programs running on blockchain

OpenSea

NFT

tether

Token

Compound

Defi

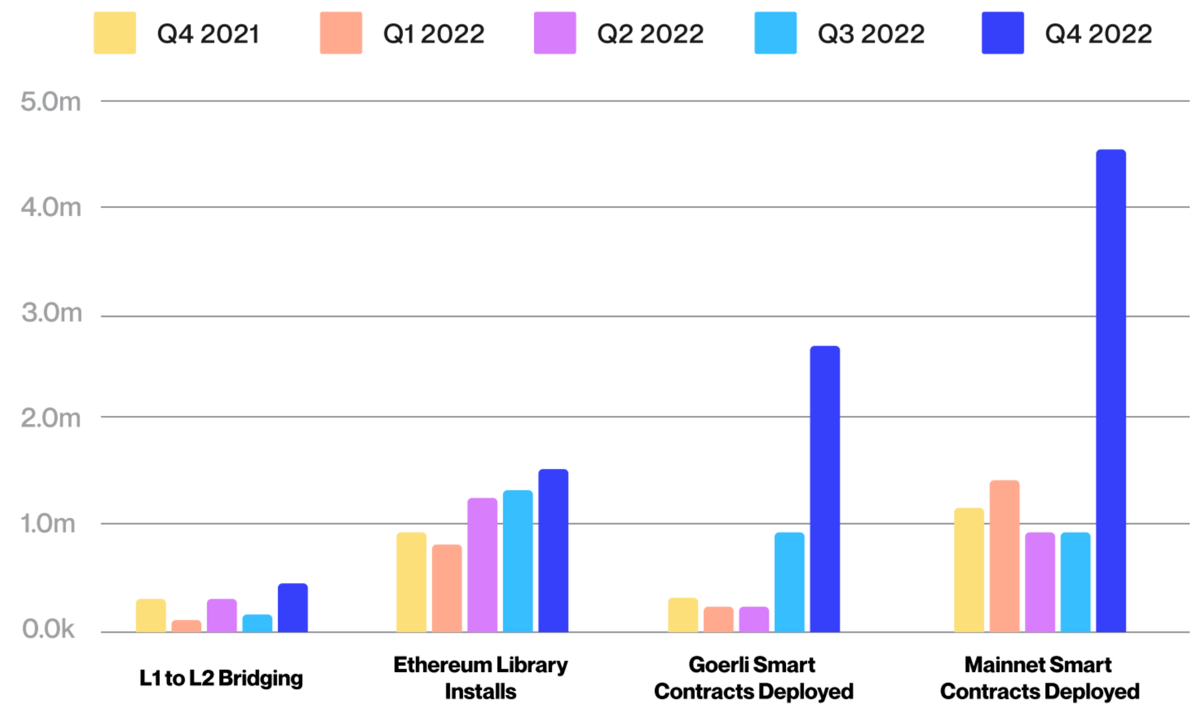
Axie INFINITY

GameFi

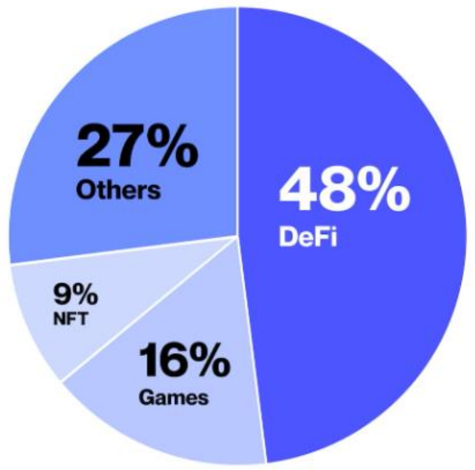
Building different decentralized apps

Smart Contract Development

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



Source: Alchemy[1]

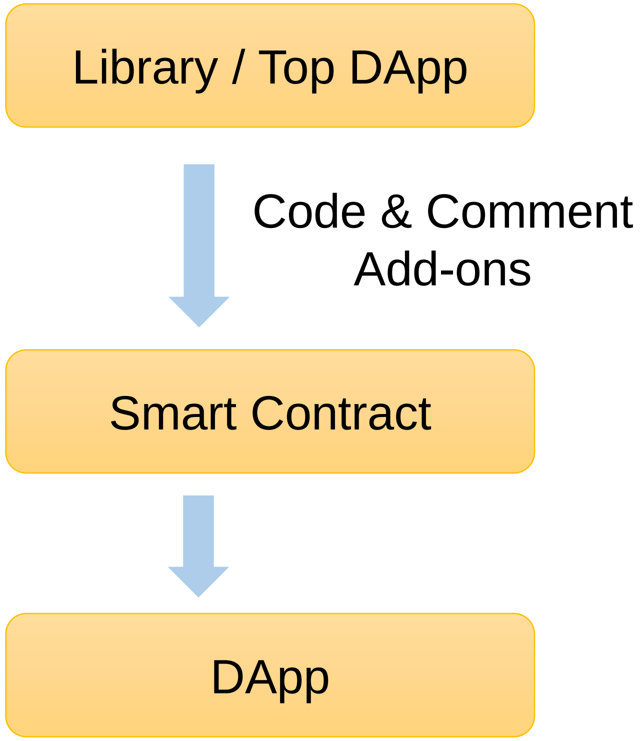


16,102 DApps in DappRadar[2]

[1] https://alchemy.com/blog/web3-developer-report-q4-2022
[2] https://dappradar.com/

Smart Contract Development

Comments are widely used and propagated in smart contracts.



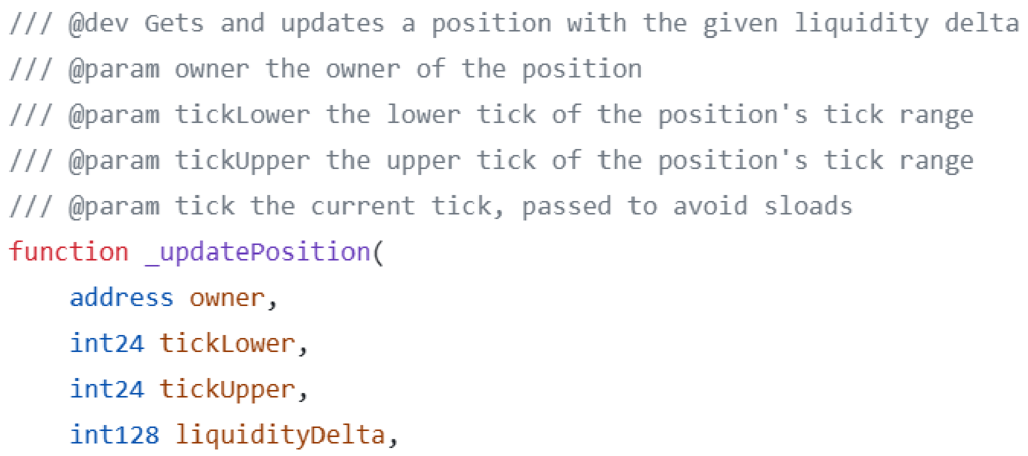
OpenZeppelin

ERC20 ERC777
ERC721 ERC1155

Library

Compound
UNISWAP

DApp



```
/// @dev Gets and updates a position with the given liquidity delta
/// @param owner the owner of the position
/// @param tickLower the lower tick of the position's tick range
/// @param tickUpper the upper tick of the position's tick range
/// @param tick the current tick, passed to avoid sloads
function _updatePosition(
    address owner,
    int24 tickLower,
    int24 tickUpper,
    int128 liquidityDelta,
```

Comment-code Inconsistency

■ Comment-code Inconsistency（CCI)

- ➢ Program code may not be perfectly aligned with comments ➡ CCI
- ➢ CCIs are highly indicative of errors in either the comments or code
- ➢ CCIs may bring confusion to app developers or end-users and even vulnerabilities

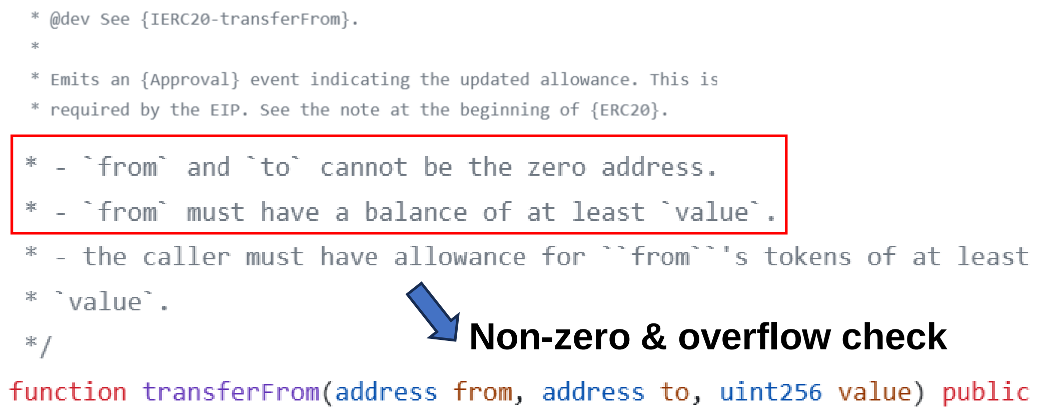
- [ICSE'11] acomment
- [ASE' 17] Fraco
- [ISSTA'18] Jdoctor
- [FSE'20] C2S
- [ASE'20] CUP
- [FSE'21] TDCleaner

Limited to specific languages or specific types of CCIs

Comment-code Inconsistency

CCIs in Smart Contract

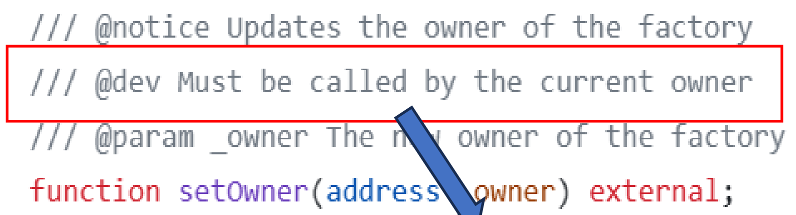
- Comments for smart contract functions **could be security-critical**



```
 * @dev See {IERC20-transferFrom}.
 *
 * Emits an {Approval} event indicating the updated allowance. This is
 * required by the EIP. See the note at the beginning of {ERC20}.
* - `from` and `to` cannot be the zero address.
* - `from` must have a balance of at least `value`.
* - the caller must have allowance for ``from``'s tokens of at least
* `value`.
*/
function transferFrom(address from, address to, uint256 value) public
```

Non-zero & overflow check

Openzeppelin Library Contract



```
/// @notice Updates the owner of the factory
/// @dev Must be called by the current owner
/// @param _owner The new owner of the factory
function setOwner(address _owner) external;
```

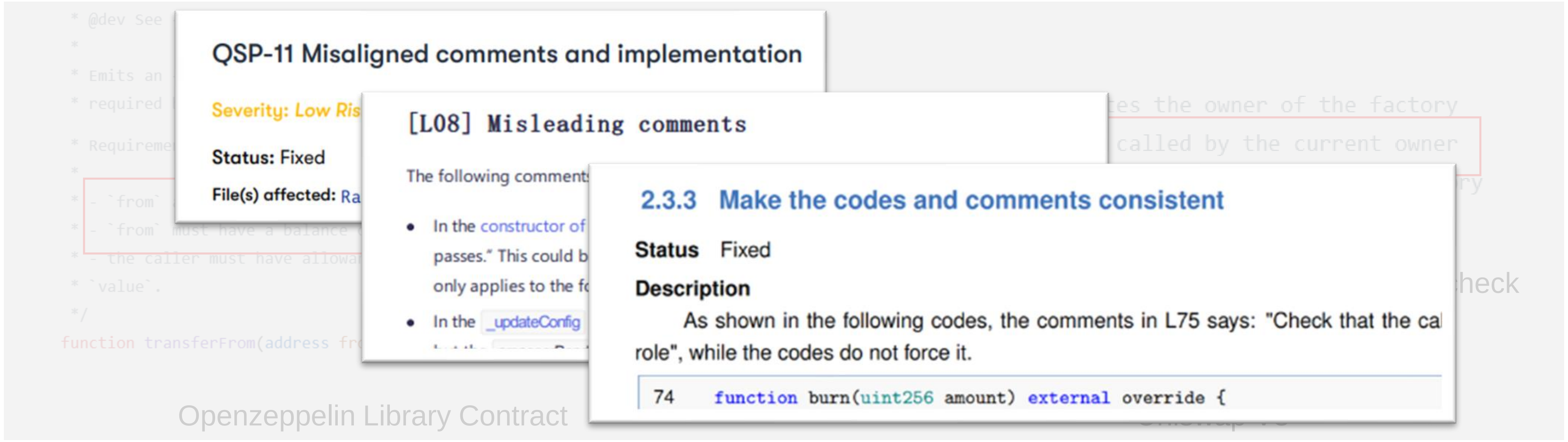
Access control check

Uniswap V3

Comment-code Inconsistency

■ CCIs in Smart Contract

➢ Audit comment-code inconsistency in smart contract



```
* @dev See
*
* Emits an
* required
* Requireme
*
* - `from`
* - `from` must have a balance
* - the caller must have allowa
* `value`.
*/
function transferFrom(address fr
```

Openzeppelin Library Contract

```
tes the owner of the factory
called by the current owner
ry
```

heck

QSP-11 Misaligned comments and implementation

Severity: Low Ris

Status: Fixed

File(s) affected: Ra

[L08] Misleading comments

The following comment

- In the constructor of
  passes." This could b
  only applies to the f
- In the _updateConfig

2.3.3 Make the codes and comments consistent

Status Fixed

Description

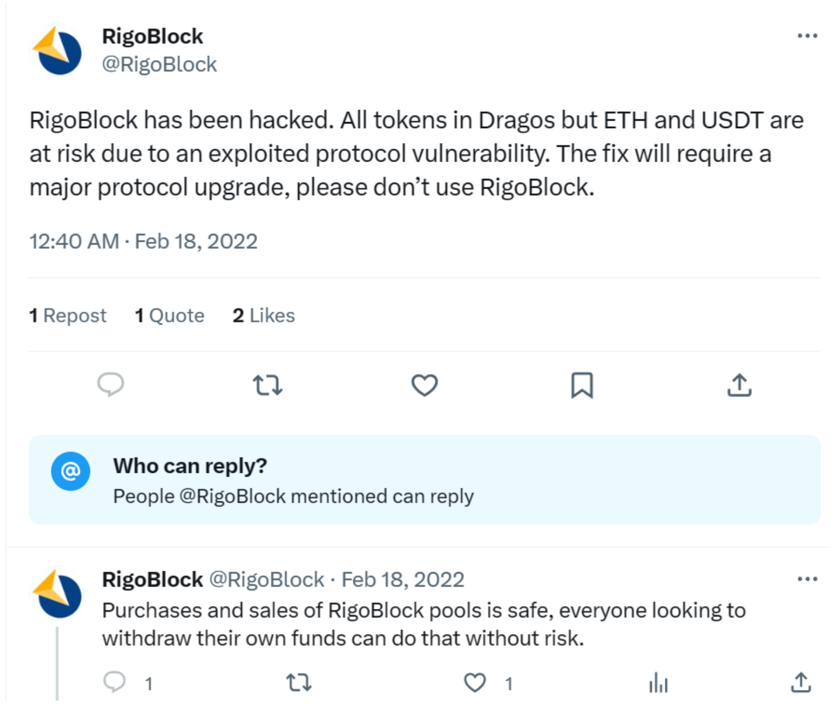
As shown in the following codes, the comments in L75 says: "Check that the cal role", while the codes do not force it.

```
74    function burn(uint256 amount) external override {
```

Such inconsistencies can cause significant losses to the contract owner and users.

Comment-code Inconsistency

Real-world Example



RigoBlock
@RigoBlock

RigoBlock has been hacked. All tokens in Dragos but ETH and USDT are at risk due to an exploited protocol vulnerability. The fix will require a major protocol upgrade, please don't use RigoBlock.

12:40 AM · Feb 18, 2022

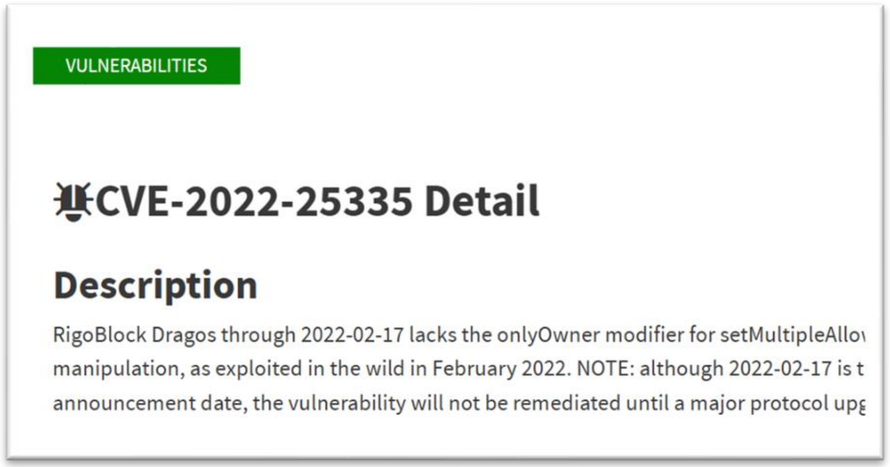
1 Repost 1 Quote 2 Likes

Who can reply?
People @RigoBlock mentioned can reply

RigoBlock @RigoBlock · Feb 18, 2022
Purchases and sales of RigoBlock pools is safe, everyone looking to withdraw their own funds can do that without risk.

1 1

Source: [1]



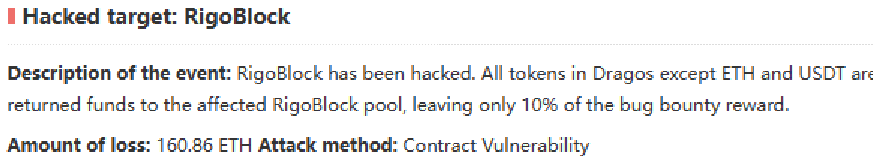
VULNERABILITIES

CVE-2022-25335 Detail

Description

RigoBlock Dragos through 2022-02-17 lacks the onlyOwner modifier for setMultipleAllo
manipulation, as exploited in the wild in February 2022. NOTE: although 2022-02-17 is t
announcement date, the vulnerability will not be remediated until a major protocol upg

Source: NVD[2]



Hacked target: RigoBlock

Description of the event: RigoBlock has been hacked. All tokens in Dragos except ETH and USDT are
returned funds to the affected RigoBlock pool, leaving only 10% of the bug bounty reward.

Amount of loss: 160.86 ETH **Attack method:** Contract Vulnerability

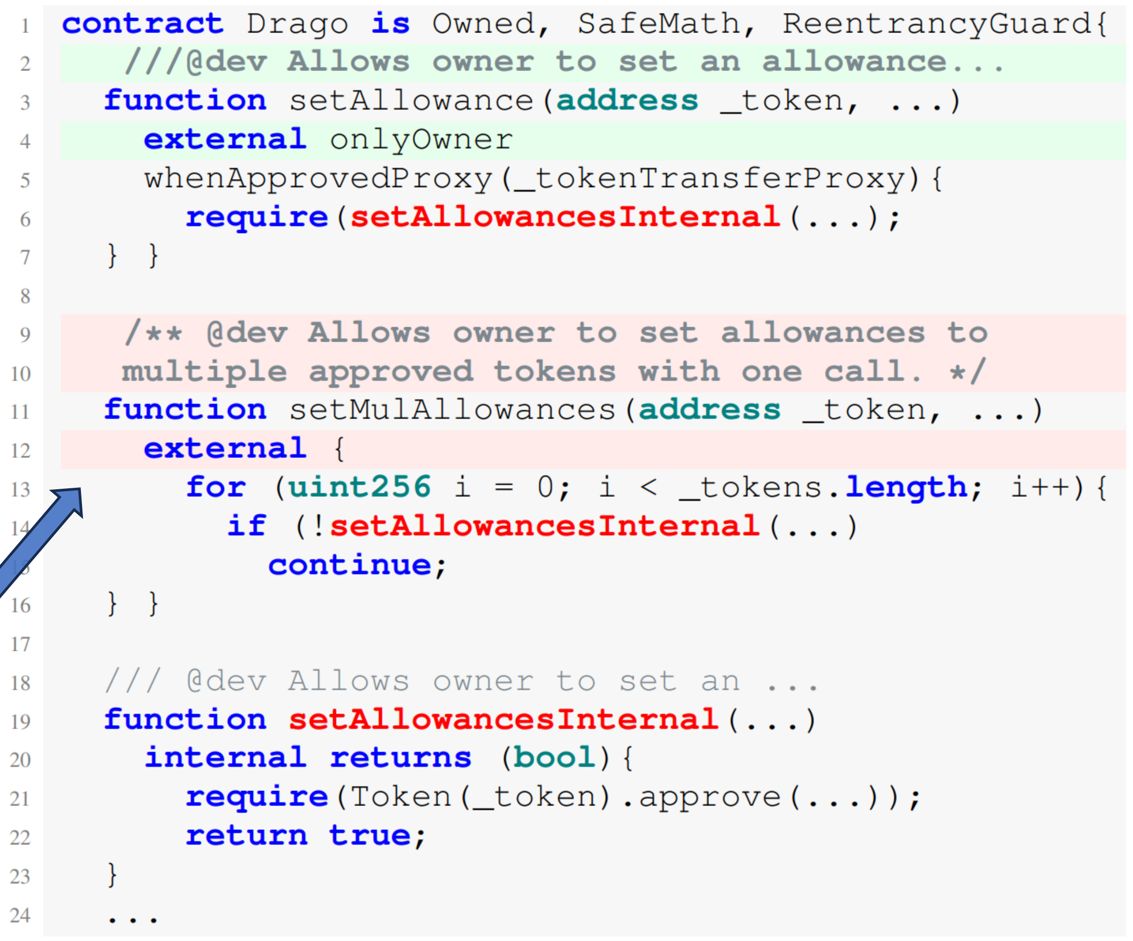
Source: SlowMist[3]

[1] https://twitter.com/RigoBlock/status/1494351180713050116
[2] https://nvd.nist.gov/vuln/detail/CVE-2022-25335
[3] https://hacked.slowmist.io/

Comment-code Inconsistency

Real-world Example

- An inconsistency of **access control**
- Green background → consistency
- Red background → inconsistency



```
contract Drago is Owned, SafeMath, ReentrancyGuard{
   ///@dev Allows owner to set an allowance...
  function setAllowance(address _token, ...)
    external onlyOwner
    whenApprovedProxy(_tokenTransferProxy){
      require(setAllowancesInternal(...);
  } }

   /** @dev Allows owner to set allowances to
   multiple approved tokens with one call. */
  function setMulAllowances(address _token, ...)
    external {
      for (uint256 i = 0; i < _tokens.length; i++){
        if (!setAllowancesInternal(...)
          continue;
  } }

  /// @dev Allows owner to set an ...
  function setAllowancesInternal(...)
    internal returns (bool){
      require(Token(_token).approve(...));
      return true;
  }
  ...
```

`setMulAllowances` only allows the owner to invoke, while the external function has no access control

Automatically reporting potential CCIs in smart contracts is in urgent need.

Problem Statement

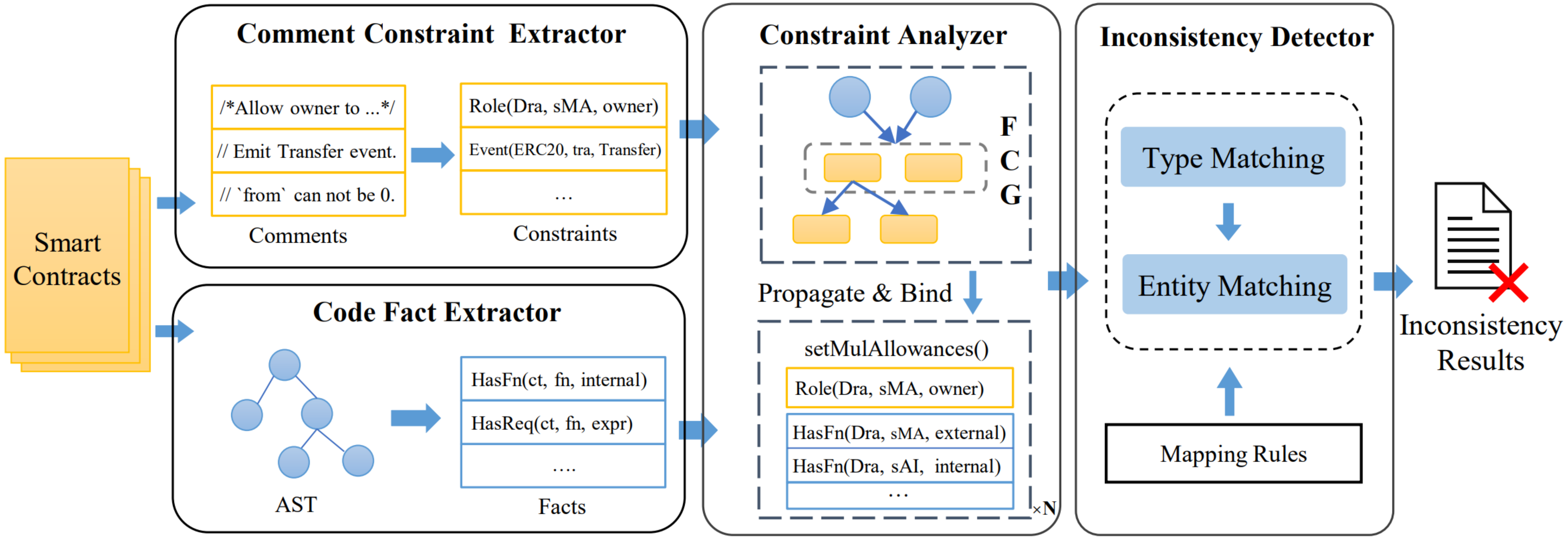
Checking CCIs in Smart Contracts

- Check CCIs in smart contracts at the function level
- We focus on three security-critical comment types

Type	Example
Role Permission	**Only available to** the **current CEO**. **Allows owner** to set allowances to multiple tokens.
Parameter Scope	from and to **cannot be** the address(0). Threshold **must be greater than** the hardcoded min.
Event Emission	**Emit** an {Approval} event. This function **emit** a {Transfer} event.

SmartCoCo

Overview



Comment Constraint Extraction

Comment Constraint Types

- Three security-critical comment types
- One additional constraint type: Comment Inheritance

Type	Constraints	Description
Role Permission	Role(c:Ct, f:Fn, role:Str)	Only **role** can invoke c.f.
Parameter Scope	Param(c:Ct, f:Fn, e:Exp)	Function c.f has a **parameter scope** with e.
Event Emission	Event(c:Ct, f:Fn, e:Str, m:Bool)	Function c.f (may) emits an e **event**.
Comment Inheritance	Inherit(sc:Ct, sf:Fn, ic:Ct, if:Fn)	Comments of sc.sf **inherits from** ic.if.

Comment Constraint Extraction

■ Comment Constraint Extraction

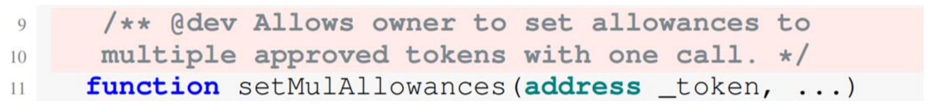
Comment preprocessing

- <contract, foo, content>
- Text transformation

Constraint templates

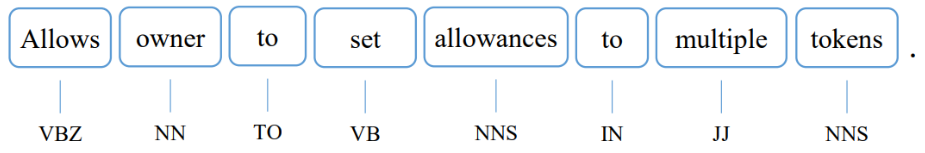
Constraint finding

- Keyword-based templates
- POS tagging filtering



```
 /** @dev Allows owner to set allowances to
 multiple approved tokens with one call. */
function setMulAllowances(address _token, ...)
```

<Drago, setMulAllowances, Allows owner to set allowances to 10 multiple approved tokens>

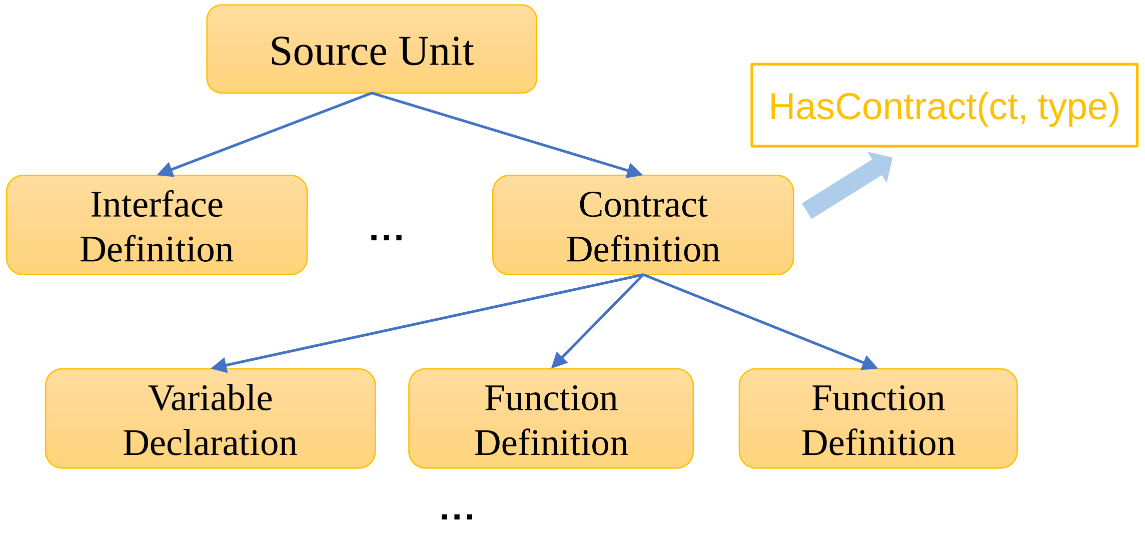


Role(Drago, setMulAllowances, owner).

Code Fact Extraction

Code Fact Definition & Extraction

- A set of code facts P(x1, ..., xn)
- Constraint -: P1(x1,…,xn), P2(x1,…,xn), …



Smart Contract AST

Code Fact	Description
HasContract(c:Ct, t:Ctype)	Contract `c` is type `t`.
HasInherit(c:Ct, ic:Ct)	Contract `c` inherits from Contract `ic`.
HasFunction(c:Ct, f:Fn, v:Vtype)	Contract `c` has a function named `f` with the visibility `v`.
FIsImplemented(c:Ct, f:Fn)	Function `c.f` has implementation.
FHasParam(c:Ct, f:Fn, p:List)	Function `c.f` has params `p`.
FHasMod(c:Ct, f:Fn, m:Fn)	Function `c.f` has modifiers `m`.
FHasEmit(c:Ct, f:Fn, e:Str)	Function `c.f` emits an event `e`.
FHasReq(c:Ct, f:Fn, e:Exp, m:Str)	Function `c.f` has a require expression `e` with an error message `m`.
FHasCall (sc:Ct, sf:Fn, a:List cc:Ct, cf:Fn, p:List)	Function `sc.sf` has a call with arguments `a` to the function `cc.cf` with parameters `p`.
Ct: Contracts in a smart contract.	Ctype ∈ {contract, interface, library}
Fn: Functions in a smart contract.	Vtype ∈ {external, public, internal, private}
List: Lists of parameters and arguments in functions and calls of a smart contract.	
Exp: Expressions in a smart contract, including arithmetic and logical expressions.	

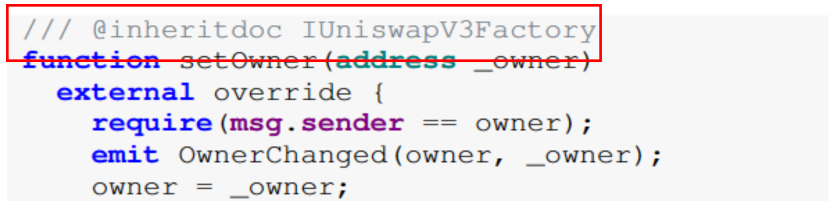
Selected code facts for further analysis

Constraint Propagation and Binding

Comment Propagation & Binding

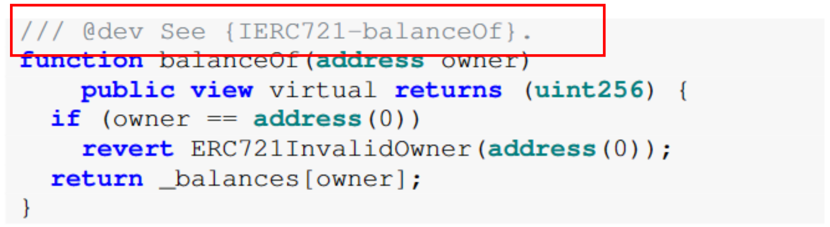
Explicit Propagation

$$Cmt(ct, fn) \; :- \; Cmt(ict, ifn),\; Inherit(ct, fn, ict, ifn)$$



```
/// @inheritdoc IUniswapV3Factory
function setOwner(address _owner)
  external override {
    require(msg.sender == owner);
    emit OwnerChanged(owner, _owner);
    owner = _owner;
```

Uniswap

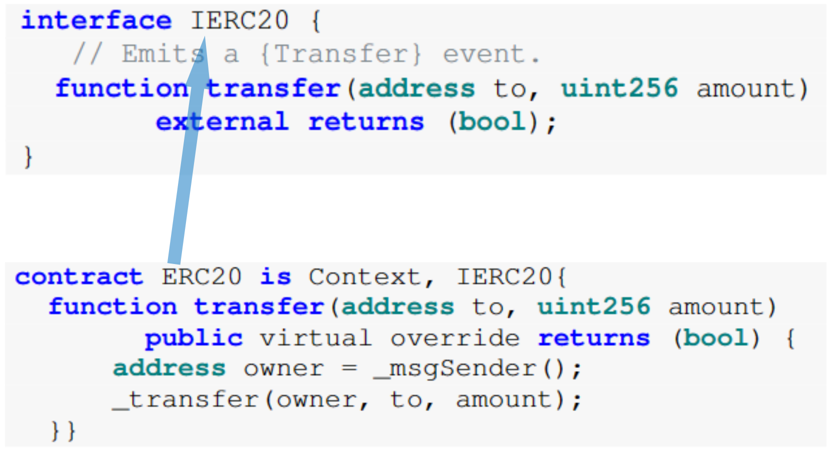


```
/// @dev See {IERC721-balanceOf}.
function balanceOf(address owner)
    public view virtual returns (uint256) {
  if (owner == address(0))
    revert ERC721InvalidOwner(address(0));
  return _balances[owner];
}
```

Token Azuki

Implicit Propagation

$$Cmt(ct, fn) \; :- \; Cmt(ict, fn),\; HasInherit(ct,\; ict),\; HasContract(ict, \texttt{interface})$$



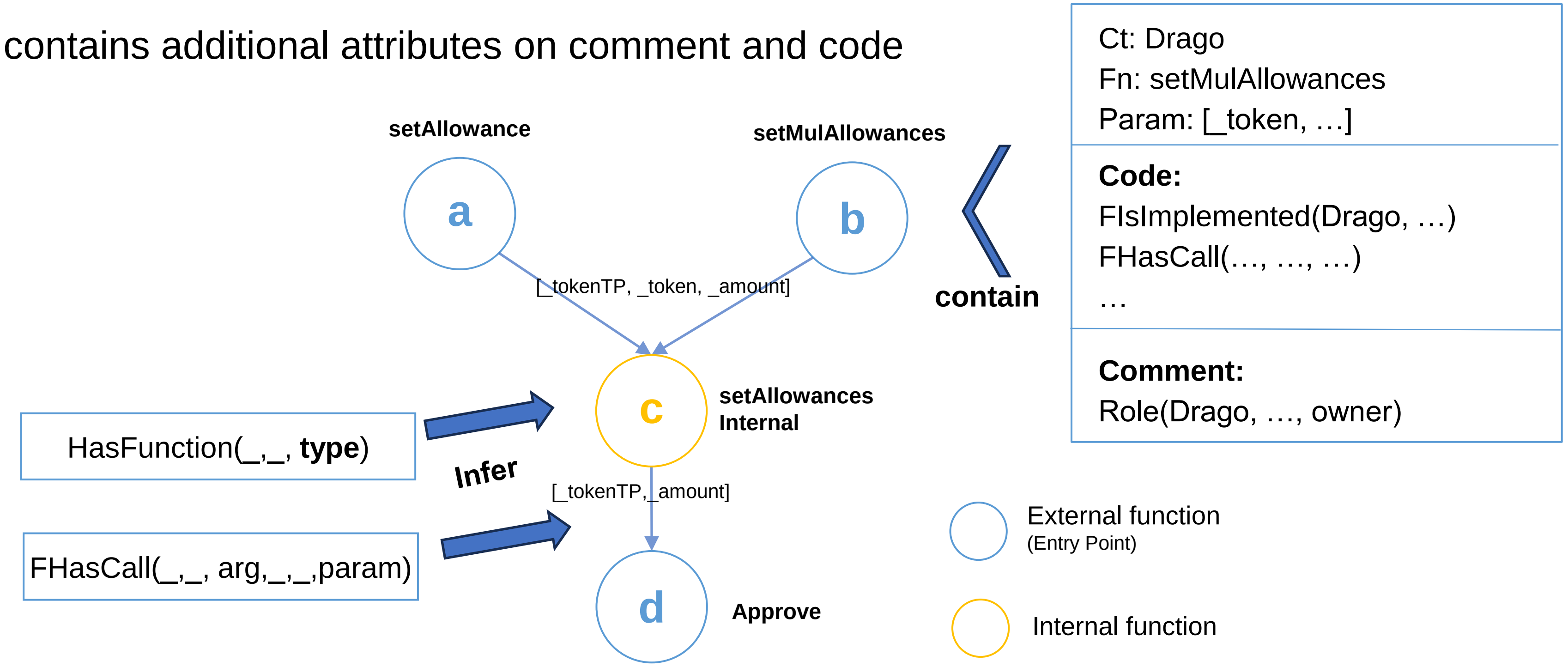
```
interface IERC20 {
   // Emits a {Transfer} event.
  function transfer(address to, uint256 amount)
        external returns (bool);
}
```

```
contract ERC20 is Context, IERC20{
  function transfer(address to, uint256 amount)
        public virtual override returns (bool) {
      address owner = _msgSender();
      _transfer(owner, to, amount);
  }}
```

Constraint Propagation and Binding

Code Propagation & Binding

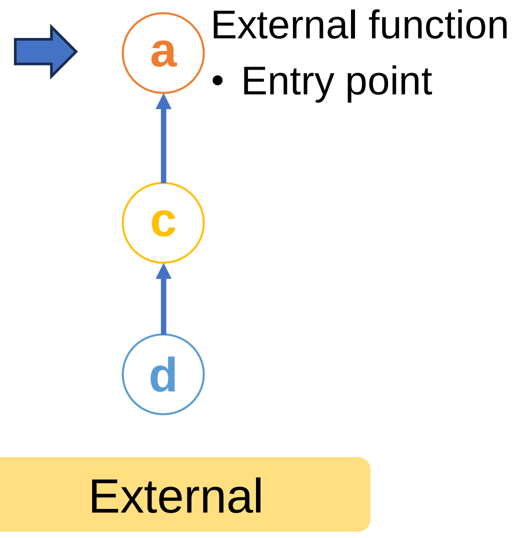
- Fact-powered call graph (FCG)
  - A subset of origin graph by eliminating functions without code facts
  - Each node contains additional attributes on comment and code

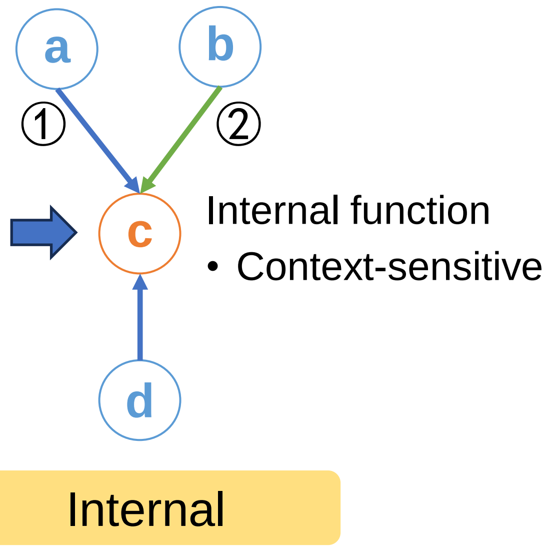


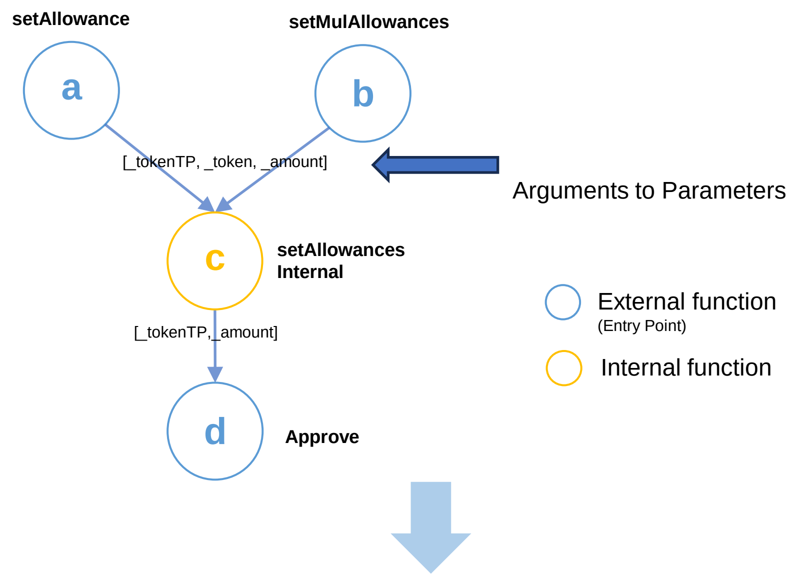
Constraint Propagation and Binding

Code Propagation & Binding

- Propagation along Call Chains







Propagated facts(PF)：
PF(a): F(a) U F(c) U F(d)
PF(c) from **a**: F(a) U F(c) U F(d)
PF(c) from **b**: F(b) U F(c) U F(d)
…

Inconsistency Detection

■ Match Constraints with Code Facts

Type Matching

- Select corresponding code facts for specific constraint

Event(_, _, Record) ➡ FHasEmit(_, _, _)

Param(_, _, a > b) ➡ FHasReq(_, _, _, _)

➡

Entity Matching

- Numbers & Expressions
  - Equivalent
- Strings
  - Similar at the character-level
  - Abbreviations

Mapping Rules

Evaluation

RQs

- **Prevalence**: What is the prevalence of proposed security-related CCIs in smart contracts?
- **Precision**: What is the effectiveness of SmartCoCo in detecting CCIs?
- **Performance**: What is the performance in checking a smart contract with constraints?

Evaluation Setup

Setup

Implementation

Python 3.10

Slither, CoreNLP, …

Conduction

20 multiple processes

Repeat experiments 3 times

Dataset

Solidity Contract

230,548 from Etherscan

139,424 unique contracts

⬇

Full Dataset

Version > 0.4.11

No compilation errors

101,780 unique contracts

Prevalence

Extracted Comments and Constraints

- 74,926 smart contracts containing 1,818,665 function comments
- SmartCoCo extracts 419,116 comment constraints in 39,372 smart contracts

Type	# Smart Contract	# Comment Constraint
Role Permission	29,963	45,725
Parameter Scope	11,582	144,653
Event Emission	21,462	137,992
Comment Inheritance	10,810	90,746
ALL	39,372	419,116

Distribution of extracted comment constraints

Prevalence

■ Identified CCIs and Distributions

➢ SmartCoCo detects 4,732 inconsistencies in 1,745 smart contracts

Type	Consistency (CCC)		Inconsistency (CCI)	
	# Smart Contract	# Instance	# Smart Contract	# Instance
Role Permission	25,951	39,781	482	697
Parameter Scope	10,940	129,171	296	507
Event Emission	14,981	122,191	995	3,528
ALL	34,639	291,143	1,745	4732

Distribution of identified CCCs and CCIs.

Precision

Precision Results

- Manually-labeled **439 unique CCIs**
- Overall, SmartCoCo achieves **a precision of 79.3%.**

Type	# CCI	# TP	# FP	Precision
Role Permission	194	145	49	74.7%
Parameter Scope	146	116	30	79.5%
Event Emission	99	87	12	87.9%
ALL	439	348	91	79.3%

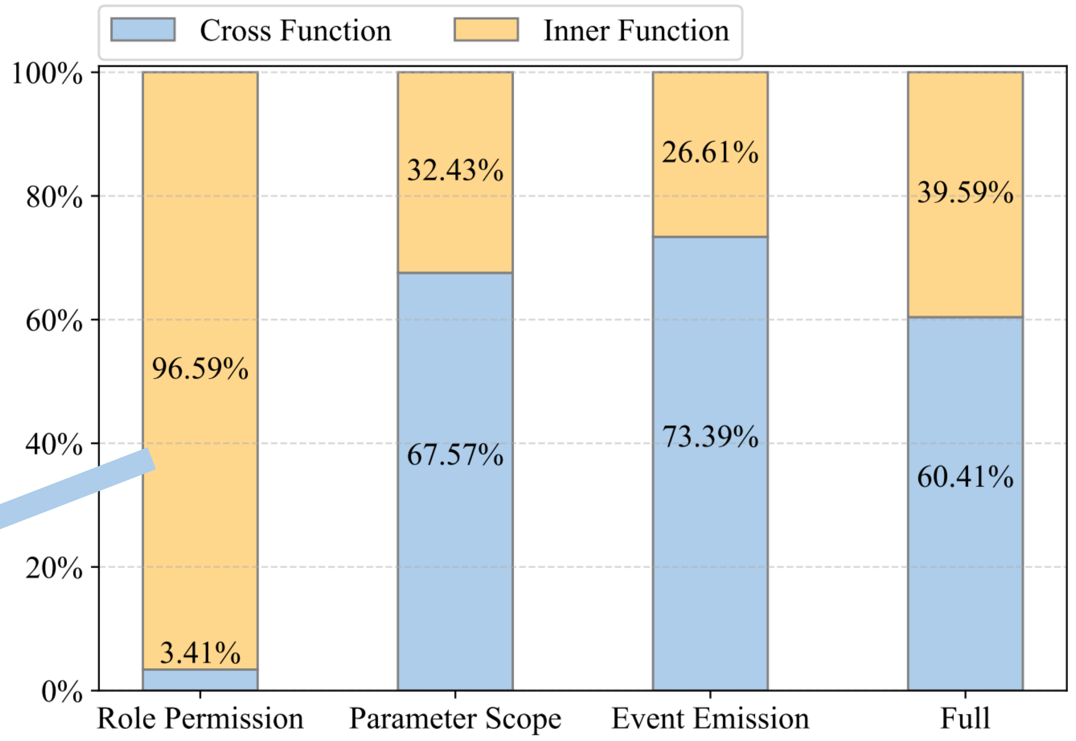
Precision of SmartCoCo over the manual-labeled CCIs.

Precision

Effectiveness of Propagation and Binding

- The position between comment and code
- **More than 60%** comment constraints implemented in another function

Modifiers account for **91.85%**



Performance

Average Analysis Time

- All contracts with different versions are successfully analyzed
- Split Full dataset → Small, Medium, and Large subsets
- SmartCoCo takes only **2.64 seconds** to analyze a contract on average.

	Small 1/3	Medium 1/3	Large 1/3	Average
Code	1.3547	2.2020	4.3698	2.6411
Comment	1.9017	2.4915	3.6671	

Detection time (in seconds)

Summary

■ Comment-code Inconsistency

- SmartCoCo presents a static framework to detect comment-code inconsistency for smart contracts with a set of propagation and binding mechanisms
- SmartCoCo reports 4,732 inconsistencies from 1,745 smart contracts, and achieves a precision of 79% on 439 manual-labeled unique inconsistencies
- SmartCoCo explores a new direction to enhance the security of smart contracts

THANKS

SmartCoCo: Checking Comment-code Inconsistency in Smart Contracts via Constraint Propagation and Binding

Email: haosch@mail2.sysu.edu.cn