Wei, C., Cai, S., Charalambous, Y., Wu, T., Godboley, S., Cordeiro, L.

Wei, C., Cai, S., Charalambous, Y., Wu, T., Godboley, S., Cordeiro, L.

MANCHESTER 1824
The University of Manchester

VeriExploit: Automatic Bug Reproduction in Smart Contracts via LLMs and Formal Methods

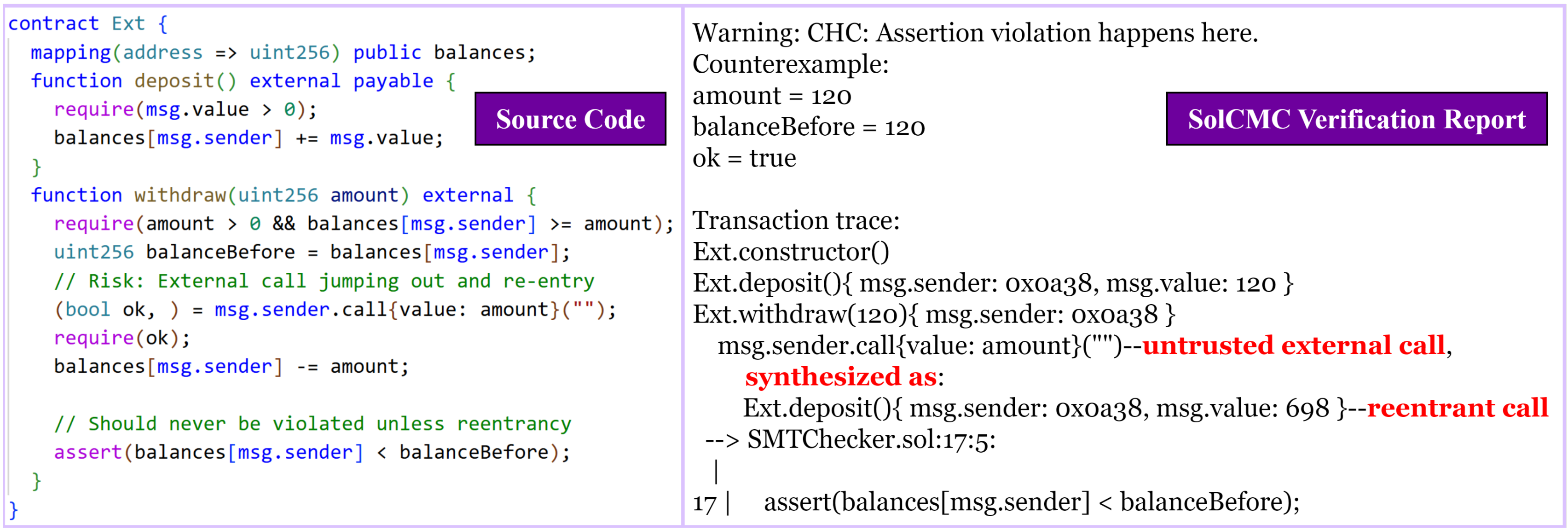
Wei, C., Cai, S., Charalambous, Y., Wu, T., Godboley, S., Cordeiro, L.

The University of Manchester; NIT Warangal

Background

- Bug finders (e.g. verifiers) **detect** issues but **rarely provide executable reproductions** or full traces.
- Many bugs are **cross-contract and multi-step** (e.g. **reentrancy**); hand-crafting attacker contracts + call sequences is slow and error-prone.
- Prior work targets **limited bug types** and lacks **formal guarantees** or **automated validation** to ensure the correctness of generated exploits.

Illustrative Example



Source Code

```
contract Ext {
  mapping(address => uint256) public balances;
  function deposit() external payable {
    require(msg.value > 0);
    balances[msg.sender] += msg.value;
  }
  function withdraw(uint256 amount) external {
    require(amount > 0 && balances[msg.sender] >= amount);
    uint256 balanceBefore = balances[msg.sender];
    // Risk: External call jumping out and re-entry
    (bool ok, ) = msg.sender.call{value: amount}("");
    require(ok);
    balances[msg.sender] -= amount;

    // Should never be violated unless reentrancy
    assert(balances[msg.sender] < balanceBefore);
  }
}
```

SolCMC Verification Report

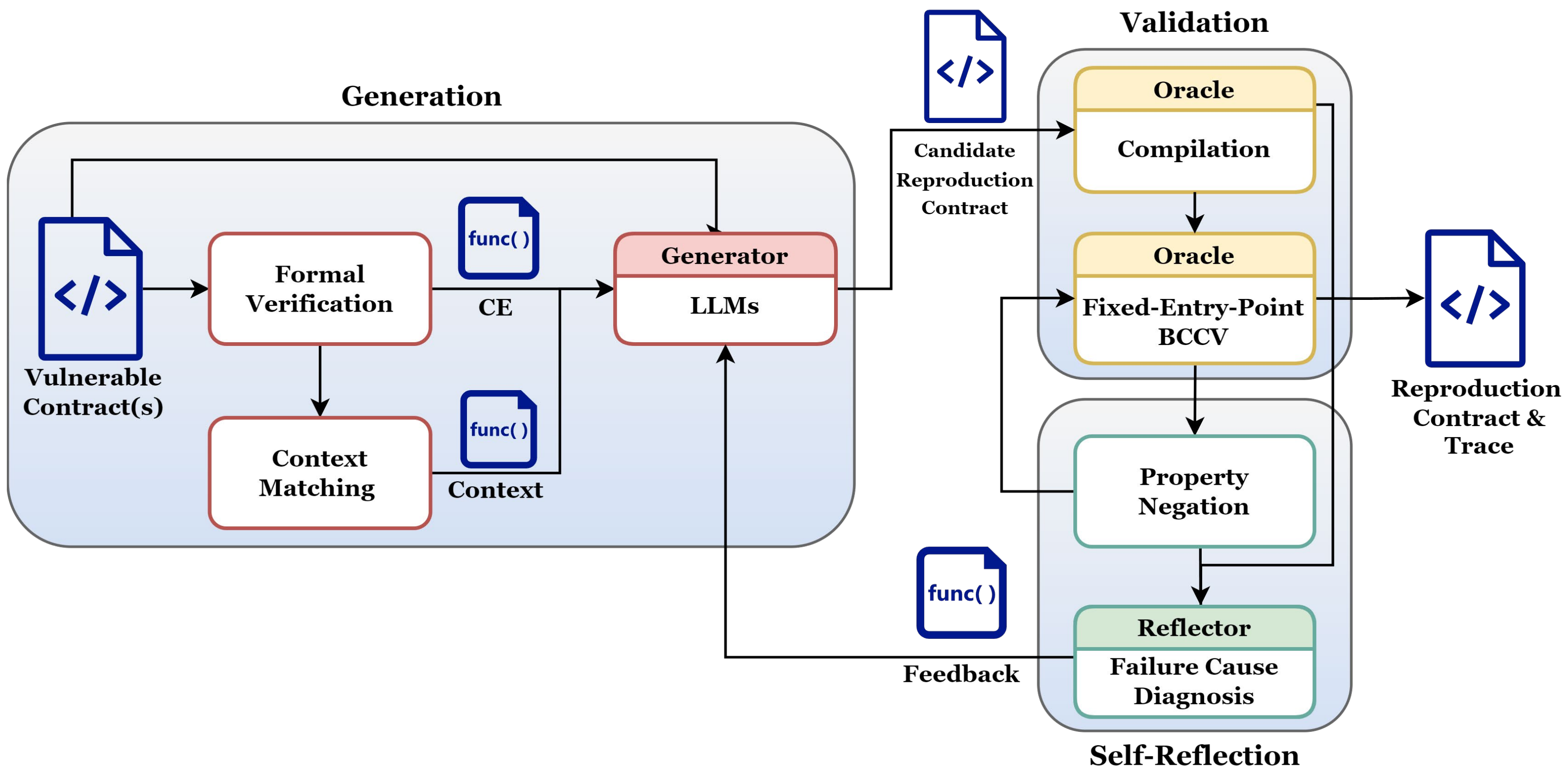
```
Warning: CHC: Assertion violation happens here.
Counterexample:
amount = 120
balanceBefore = 120
ok = true

Transaction trace:
Ext.constructor()
Ext.deposit(){ msg.sender: 0x0a38, msg.value: 120 }
Ext.withdraw(120){ msg.sender: 0x0a38 }
  msg.sender.call{value: amount}("")--untrusted external call,
    synthesized as:
    Ext.deposit(){ msg.sender: 0x0a38, msg.value: 698 }--reentrant call
 --> SMTChecker.sol:17:5:
  |
17 |    assert(balances[msg.sender] < balanceBefore);
```

- The external reentrant call is "**synthesized**" and therefore incomplete.
- Even if reported as a "**reentrant call**", it does not guarantee that the bug truly results from reentrancy.

Proposed Methods

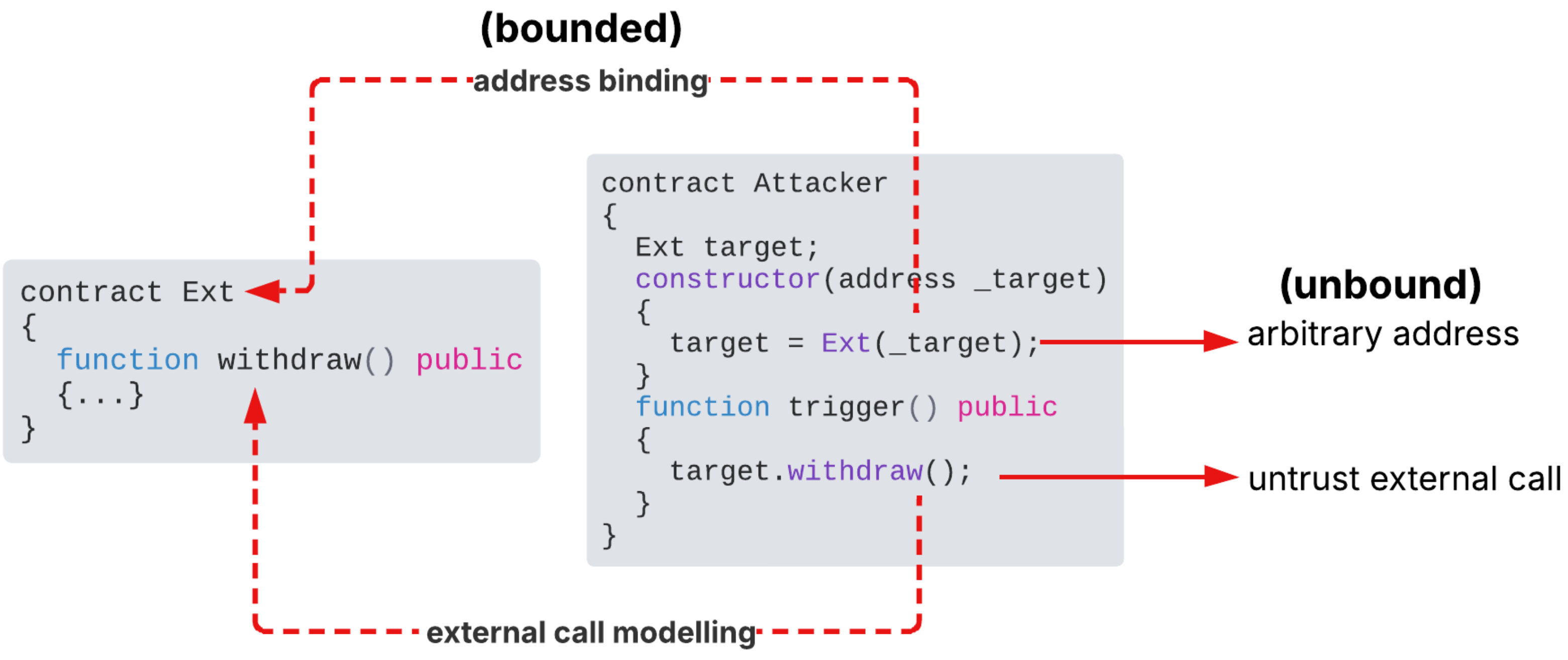
VeriExploit: "LLM-guided generation" + "BCCV verification".

- Generalizes to multiple vulnerability types.
- Formally and automatically validates exploits (rather than manual inspection or exhaustive dynamic traversing).
- Enriches verifier counterexample traces.



Bounded Cross-Contract Verification (BCCV)

Given a contract system containing vulnerable and attacker contracts, the model of BCCV is to formally validate if there exists an exploit.



```
contract Ext
{
  function withdraw() public
  {...}
}
```

```
contract Attacker
{
  Ext target;
  constructor(address _target)
  {
    target = Ext(_target);
  }
  function trigger() public
  {
    target.withdraw();
  }
}
```

LLM Generation/Reflection

- **Generation**: an additional context of vulnerability patterns and traces is added to guide faithful exploit synthesis.
- **Self-reflection**: a property-negation strategy is introduced to diagnose failures and suggest targeted fixes.

Implementation

- We build VeriExploit upon two off-the-shelf tools, **SolCMC** (SMTChecker) and **ESBMC**.
- Both verifiers naturally support **unbounded reasoning**, multiple properties, and trace generation.

Results

Comparison of Success Rate

	Baseline	SOLCMC	ESBMC
AO	7.65%	74.12%	88.24%
AU	23.33%	55.56%	74.44%
DZ	28.57%	71.43%	100.00%
OB	14.29%	85.71%	85.71%
AV	1.82%	74.55%	98.18%
RE	7.83%	44.35%	80.00%
Tot.	11.80%	64.60%	85.60%

(AO: Arithmetic Overflow, AU: Access Control, DZ: Division by Zero, OB: Out-of-Bounds, AV: Assertion Violation, RE: Reentrancy)

VeriExploit is effective across diverse vulnerability types

VeriExploit shows scalability even on larger contracts

Scalability by size strata. **Simp.** = SLOC $\leq$ dataset median; **Comp.** = SLOC $>$ dataset median. SC = success rate (%); RT = average runtime (s); IT = average reflection iterations.

	SC (%)		RT (s)		IT	
	Simp	Comp	Simp	Comp	Simp	Comp
SolCMC	77.20	52.00	25.41	73.88	1.12	1.06
ESBMC	90.00	81.20	23.74	45.78	0.91	1.33

VeriExploit performs on par with prior work on reentrancy

Comparison of success rate on AdvScanner dataset

AdvScanner (reported)	VeriExploit (ESBMC)
88.48%	90.91%

Conclusion and Ongoing Works

- VeriExploit combines formal verification and LLMs to reproduce smart-contract vulnerabilities.
- We plan to use partial generation to reduce cost and improve scalability.
- We will explore state-aware reproduction for real-world on-chain contracts.

This work was supported by XC5 Hong Kong Limited (CR No. 2647858).

Department of Computer Science, UoM **www.cs.manchester.ac.uk**

🎥 **Watch our teaser: youtu.be/b9X7AAzeSh4**