PoCo: Agentic PoC Exploit Generation for Smart Contracts

Vivi Andersson*, **Sofia Bobadilla***, Harald Hobbelhagen, Martin Monperrus

Accepted @ ACM TOSEM Special issue on Agentic AI in Software

Vivi Andersson <vivia@kth.se>

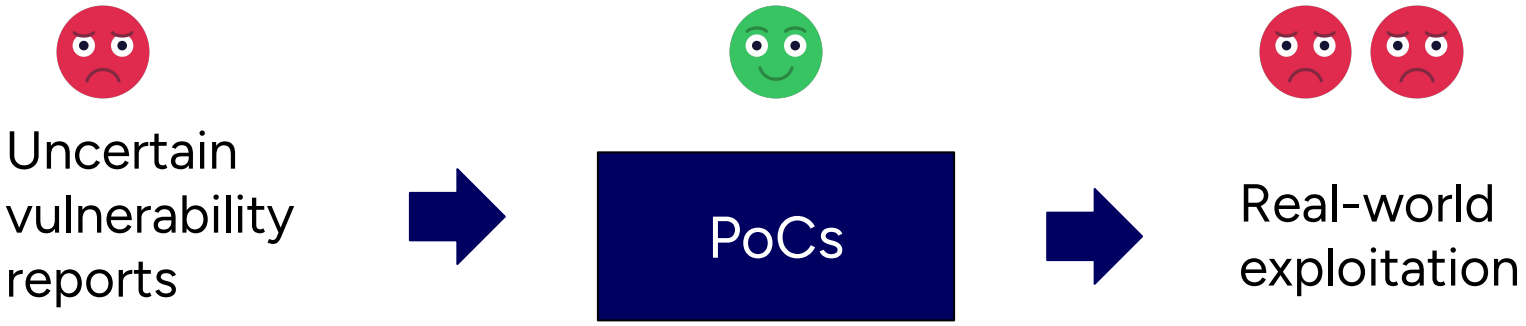
Dagstuhl, Evaluation of AI Models in SE

2026-05-06

How we evaluated an agentic exploit generator for smart contracts

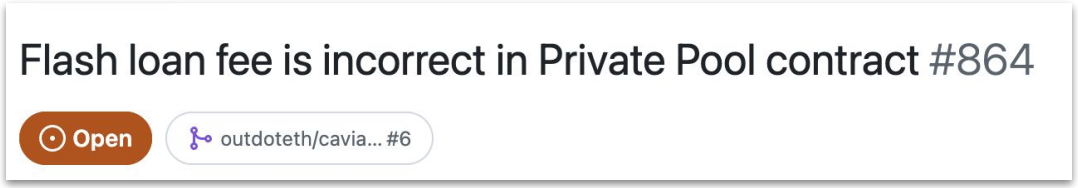
Motivation

Why PoCs?



A Proof-of-Concept is **executable evidence** of exploitability

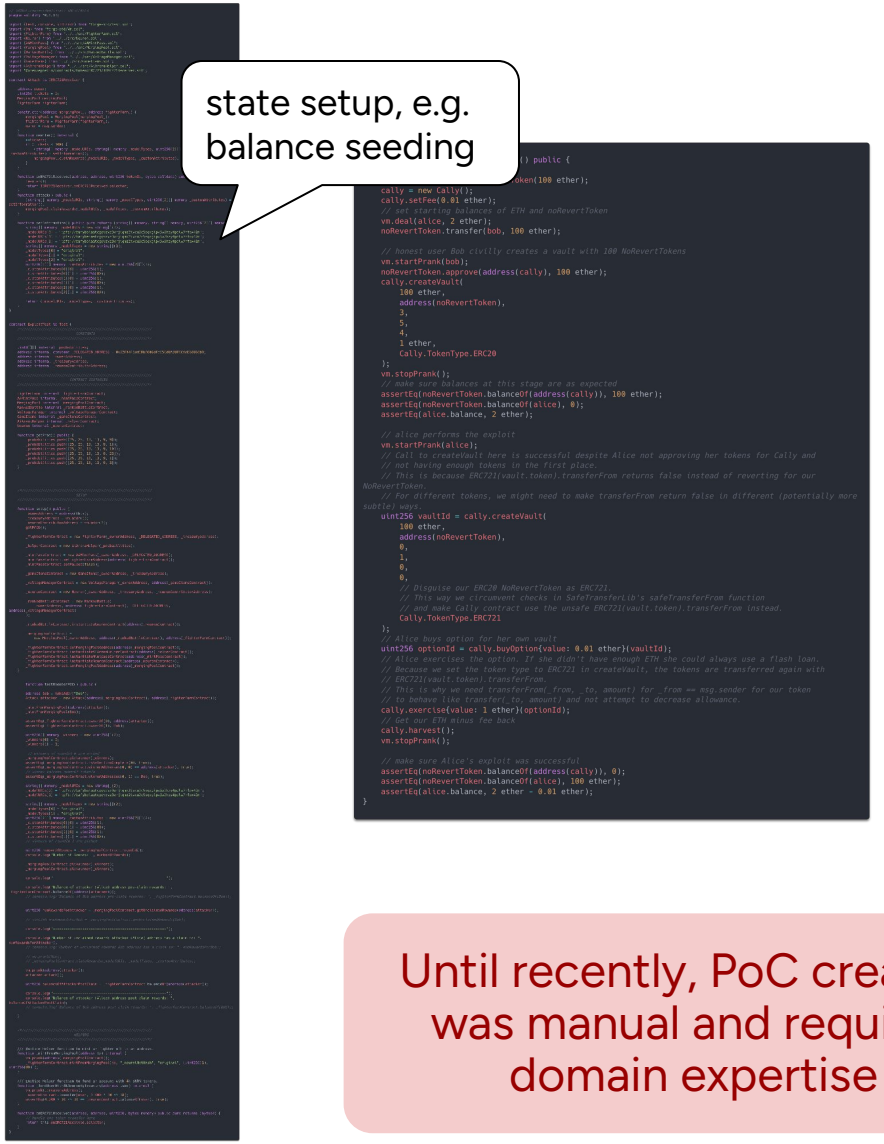
Smart Contract PoCs

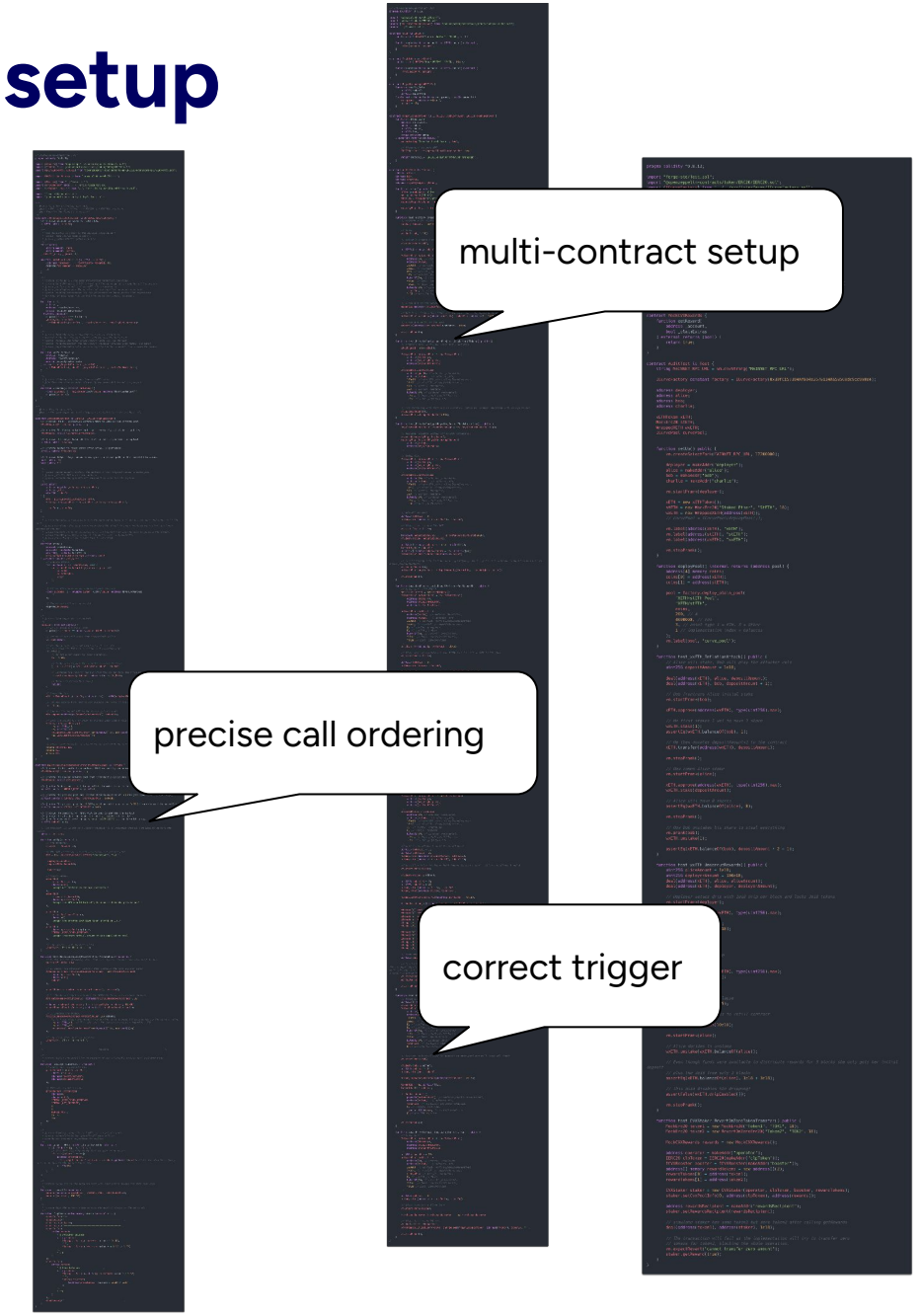


1. **Set up:** deploy contracts, fund wallets, configure exploitable state
2. **Trigger:** execute bad sequence exploiting a flaw
3. **Assert:** verify a security violation occurred

```
function test_flashLoanFeeExploit() public {
    // ① SET UP
    PrivatePool pool = new PrivatePool();
    pool.initialize(address(nft), changeFee);
    vm.deal(alice, 1 ether);

    // ② TRIGGER
    vm.prank(alice);
    pool.flashLoan(borrower, tokenId, "");

    // ③ ASSERT fee paid is 25 wei,
    // not 25×10^14
    assertEq(feePaid, 25,
        "fee should be 0.0025 ETH, got 25 wei");
    assertLt(feePaid, expectedFee);
}
```

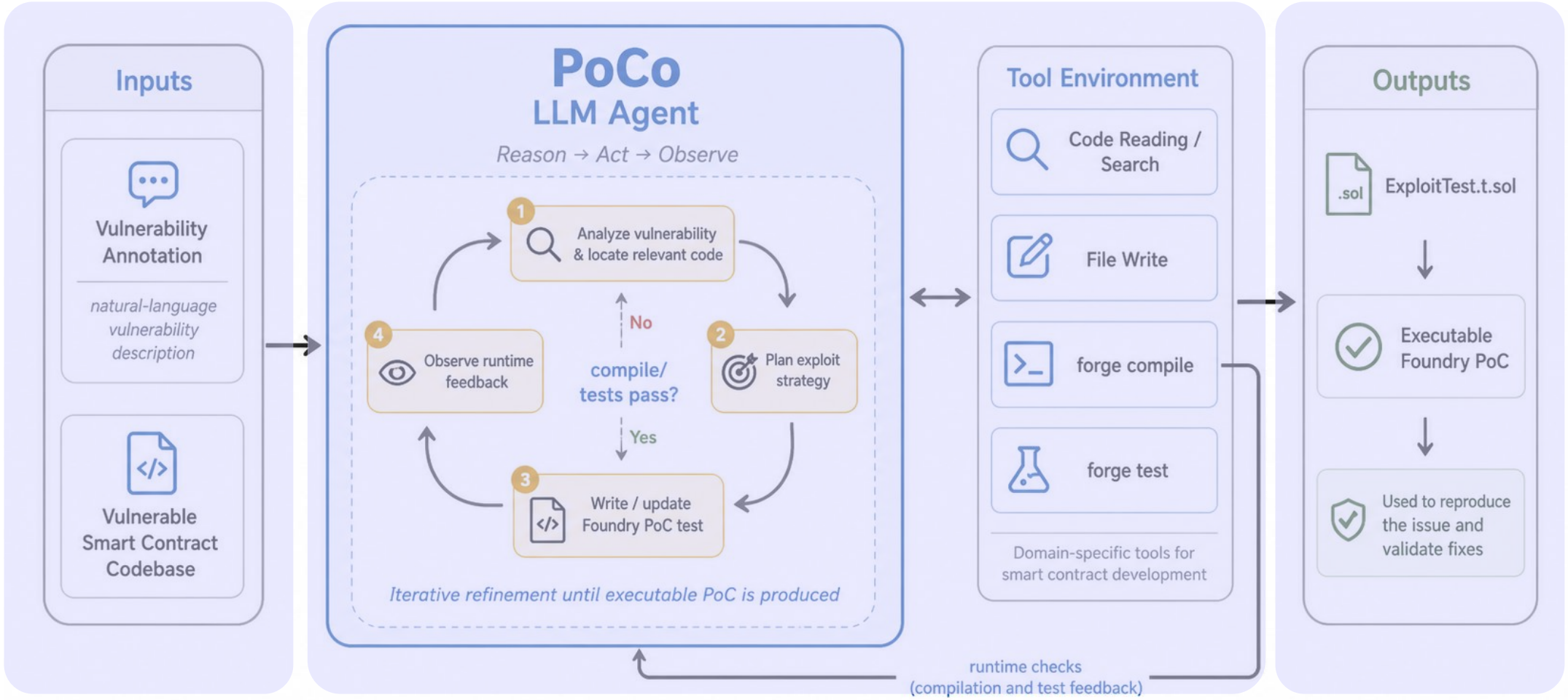
Real PoCs require substantial setup





Until recently, PoC creation was manual and requires domain expertise

PoCo: Vulnerability Report to Exploit



Runtime oracle = assertion success

..but passing tests are not correctness!

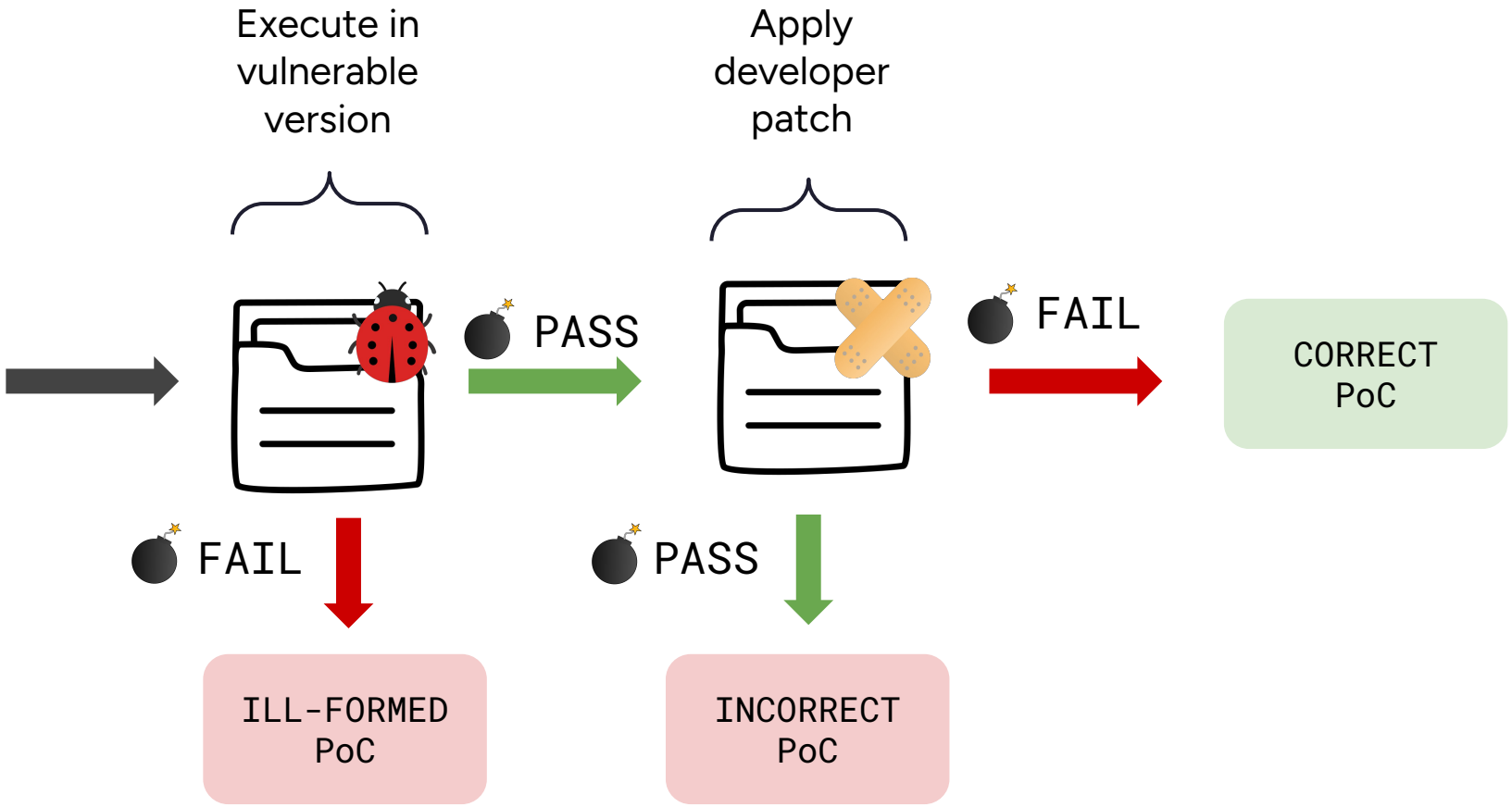
```
assertTrue(true);                  // meaningless
assertEq(balance, balance);        // also meaningless
assertEq("super mocked", bad)      // not a real exploit
```

How to faithfully evaluate logical correctness of generated PoCs?

Patches as a correctness oracle

A post-hoc patch-based evaluation

```
contract Exploit {
  function run() external {
    IVault(target).withdraw(
      type(uint256).max);
  }
}
```



Proof-of-Patch

23 manually curated real-world smart contract vulnerabilities

Each sample contains:

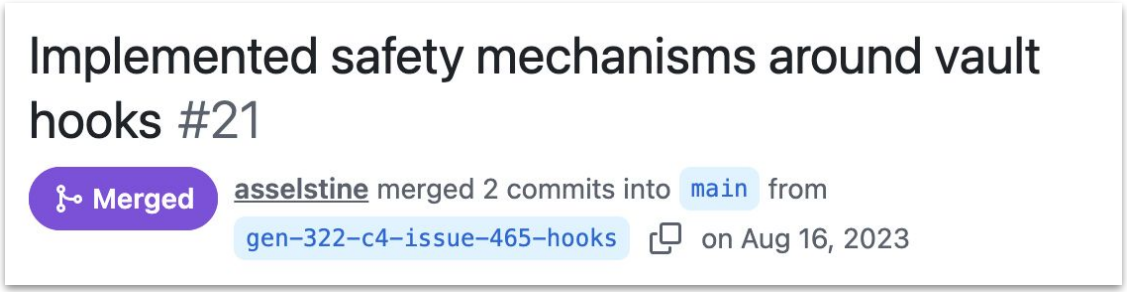
1. Vulnerability description
2. Vulnerable codebase
3. Minimal patch

ID	Project	Description
001	2024-06-size	Logical error in multicall function allows users to bypass deposit limits.
003	2023-07-pooltogether	User can mint shares to any address and steal the yield fee of the protocol.
008	2023-09-centrifuge	Rounding errors in share calculations allow investors to receive excess shares.
009	2023-04-caviar	Royalties are miscalculated when recipient address is zero, leading to trapped funds.
015	2023-07-pooltogether	The prize-winners hook mechanism can be exploited to interfere with the intended prize distribution process.
018	2023-04-caviar	Former owner can set token approvals that enable them to reclaim assets after ownership transfer.
020	2023-12-dodo-gsp	A first liquidity provider can inflate the share price during pool initialization, enabling a DoS.
032	2022-06-putty	User cannot withdraw their strike amount and their asset will be stuck in the contract.
033	2023-04-caviar	The PrivatePool contract miscalculates flash loan fees causing incorrect fee totals.
039	2024-03-axis-finance	Refund handling errors can lock seller funds when the token reverts on zero transfers.
041	2024-03-axis-finance	User can hijack a prefunded auction and gain control over its deposited funds.
042	2025-07-cap	User can exploit a rounding error to repeatedly miscompute utilization, causing inaccurate interest rate adjustments.
046	2023-05-xeth	Zero token transfer can cause a potential denial of service when giving rewards
048	2023-04-caviar	Malicious royalty recipient can extract value from the pool without proper payment.
049	2023-08-cooler	Lender can update loan terms without borrower approval, enabling them to impose unfair conditions.
051	2023-09-centrifuge	Missed access control allows users to deposit on behalf of others and potentially caused a denial of service attack.
054	2022-05-cally	Unchecked token transfer return values let attackers create empty vaults, causing buyers to pay Ether but receive no tokens.
058	2022-06-putty	Users can accidentally send Ether to code paths that don't use it, causing the funds to be locked
066	2023-11-kelp	Users receive less rsETH than expected due to a miscalculation in the minting logic.
070	2024-08-ph	Users are able to transfer NFT tokens even when the contract is paused.
077	2024-02-ai-arena	Players can exploit a reentrancy bug to claim extra rewards before the contract updates their NFT balance.
091	2023-07-basin	Users can manipulate the reported asset reserves, causing incorrect price data.
098	2022-05-cally	Fake token balances can be created for nonexistent ERC20s, enabling traps that steal funds from later users.
Total	**23**	

Why do we trust the patches?

1. DEVELOPER-ACCEPTED

- Created / reviewed by core devs
- Merged as the security fix



2. MINIMAL SECURITY FIXES

- Isolate vulnerability fixes
- No arbitrary bundled changes

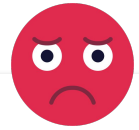
Curated Security Patches

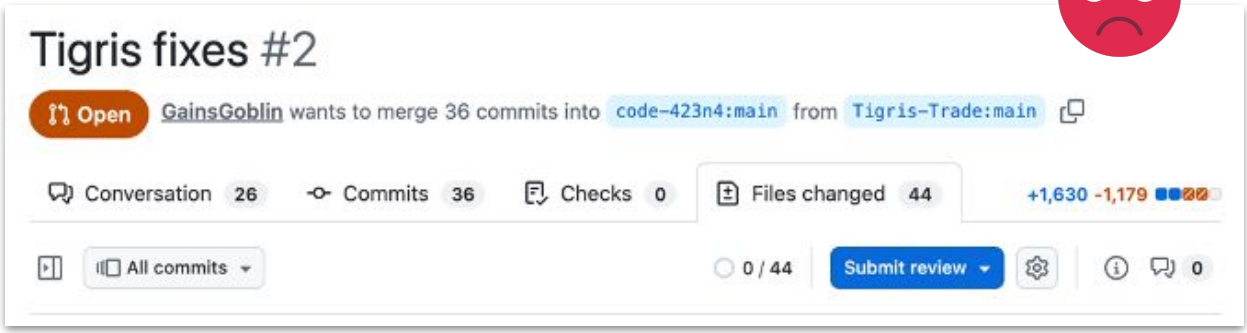
```
--- a/2023-11-kelp/src/LRTDepositPool.sol
+++ b/2023-11-kelp/src/LRTDepositPool.sol
@@ -125,6 +125,11 @@
         nonReentrant
         onlySupportedAsset(asset)
     {
+
+        // interactions
+        uint256 rsethAmountMinted = _mintRsETH(asset, depositAmount);
+
+
         // checks
         if (depositAmount == 0) {
             revert InvalidAmount();
@@ -137,8 +142,6 @@
             revert TokenTransferFailed();
         }

-        // interactions
-        uint256 rsethAmountMinted = _mintRsETH(asset, depositAmount);

         emit AssetDeposit(asset, depositAmount, rsethAmountMinted);
     }
```

Moves function call before a transfer

```
--- a/2022-05-cally/contracts/src/Cally.sol
+++ b/2022-05-cally/contracts/src/Cally.sol
@@ -169,6 +169,7 @@
         require(dutchAuctionReserveStrike < strikeOptions[dutchAuctionStartingStrikeIndex],
"Reserve strike too small");
         require(durationDays > 0, "durationDays too small");
         require(tokenType == TokenType.ERC721 || tokenType == TokenType.ERC20, "Invalid
token type");
+        require(token.code.length > 0, "token is not contract");

         Vault memory vault = Vault({
             tokenIdOrAmount: tokenIdOrAmount,
```

Adds invariant check to prevent fake asset

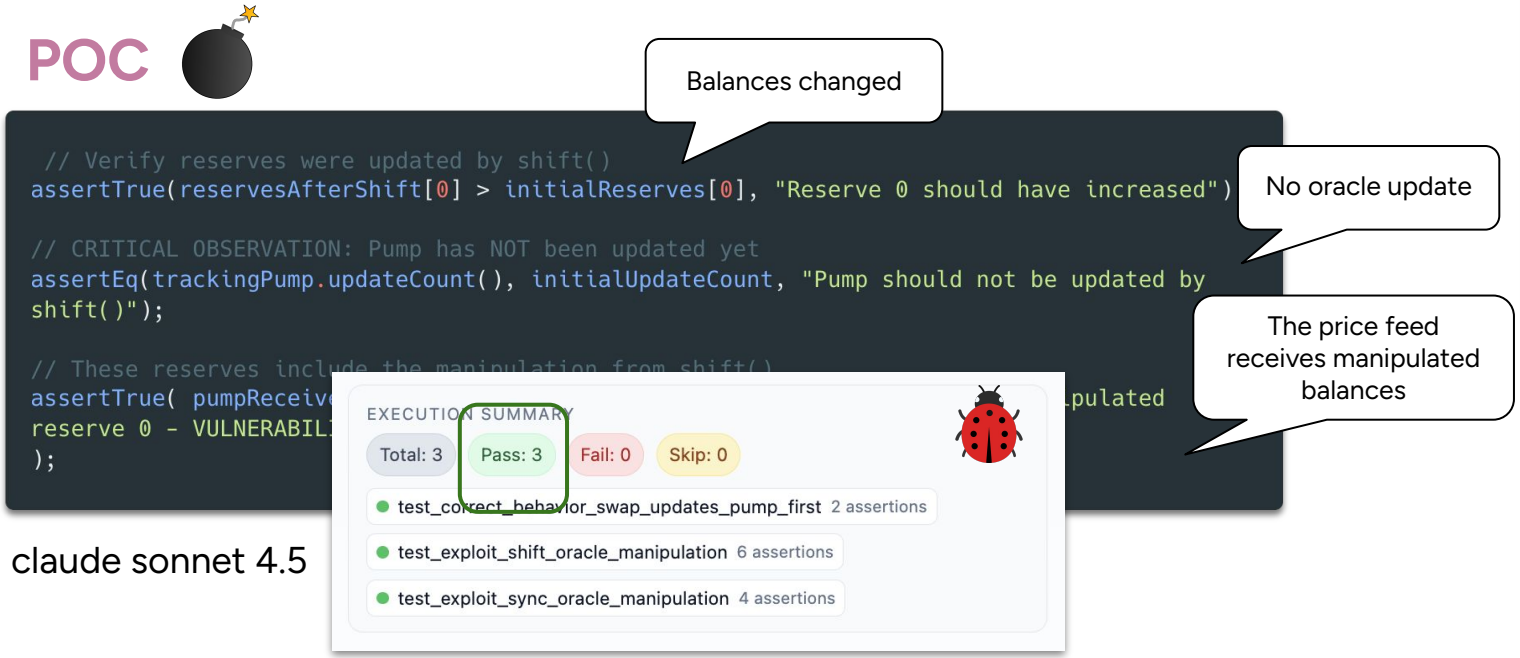




Price Oracle Manipulation (Basin #091)

- **Vulnerability:** shift() changes Basin's asset balances without updating its price feed
- **Exploit:** An attacker can manipulate Basin's price data and steal funds from protocols that trust it

POC



claude sonnet 4.5

PATCH

Evaluation Results

In short

Baselines

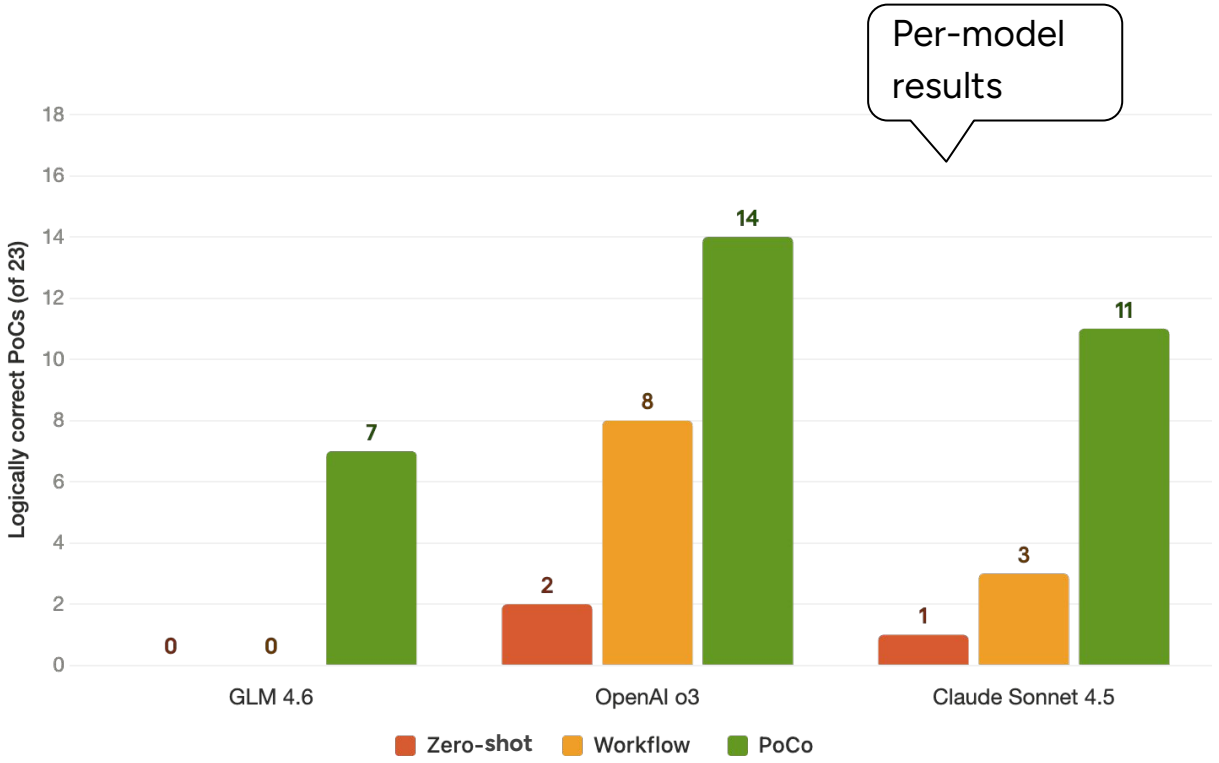
- **Zero-shot LLM:** single prompt, no iteration, no tool access
- **Workflow LLM:** LLM in a fixed, predetermined loop (compile, fix, test)

Models

- **Claude Sonnet 4.5** (frontier coding)
- **OpenAI o3** (reasoning)
- **GLM 4.6** (open weights)

Resource limits: maximum **$3** per task, or **10** smart contract tool calls

Logical Correctness



Solved cases (union across models):

- Zero-shot: 2 / 23
- Workflow: 9 / 23
- **PoCo:** 17 / 23

PoCo improves logical correctness across all models over non-agentic baselines

Manual Evaluation

Did patch-failing PoCs exploit the same bug as auditor PoCs?

- **Result:** 6 / 6 matched auditor exploit logic
- **Failure:** (1) exploit **assertion** no longer holds, or (2) exploit **operation** is no longer possible

ID	Project	#Tests	Verdict	PoC Test Outcome (✗ / ✓ on patch)	Exploit Assertion/Reverted Call
020	2023-12-dodo-gsp	1	Correct	✗ test_buyShares_lowLiquidity	gsp.buyShares(attacker)
033	2023-04-caviar	5	Correct	✗ test_FlashLoanFeeIncorrectlyScaled	assertEq(expectedFlashFee/actualFlashFee,10**14)
			Correct	✗ test_ChangeFeeVsFlashFeeDiscrepancy	assertEq(flashFeeAmount, 25)
			Correct	✗ test_EconomicImpactOfVulnerability	assertEq(actuallyPays, 2500)
			Correct	✗ test_ExploitFlashLoanWithMinimalFee	assertEq(feeReceived, 25)
			Correct	✗ test_FlashLoanFeeIncorrectlyScaledWithERC20	assertEq(actualFee, 25)
041	2024-03-axis-finance	2	N/A	✓ test_RootCause_LotIdInitializedToZero	N/A
			Correct	✗ test_StealPrefundedTokensByOverwritingLot...	assertEq(seller0After, ATTACKER)
046	2023-05-xeth	3	Correct	✗ test_getReward_RevertsOnZeroBalance	vm.expectRevert("RevertOnZeroToken: zero amount")
			Correct	✗ test_getReward_BlocksLegitimateRewards	vm.expectRevert("RevertOnZeroToken: zero amount")
			Correct	✗ test_getReward_PermanentDoS	vm.expectRevert("RevertOnZeroToken: zero amount")
070	2024-08-ph	2	Correct	✗ test_PauseDoesNotPreventTransfers	nft.safeTransferFrom(attacker, victim, tokenId, 1, "");
			Correct	✗ test_PauseDoesNotPreventBatchTransfers	nft.safeBatchTransferFrom(.., .., tokenIds, amounts, "")
091	2023-07-basin	3	N/A	✓ test_correct_behavior_swap_updates_pump_first	N/A
			Correct	✗ test_exploit_sync_oracle_manipulation	assertEq(pump.updateCount(), initialUpdateCount + 1)
			Correct	✗ test_exploit_shift_oracle_manipulation	assertEq(pump.updateCount(), initialUpdateCount + 1)

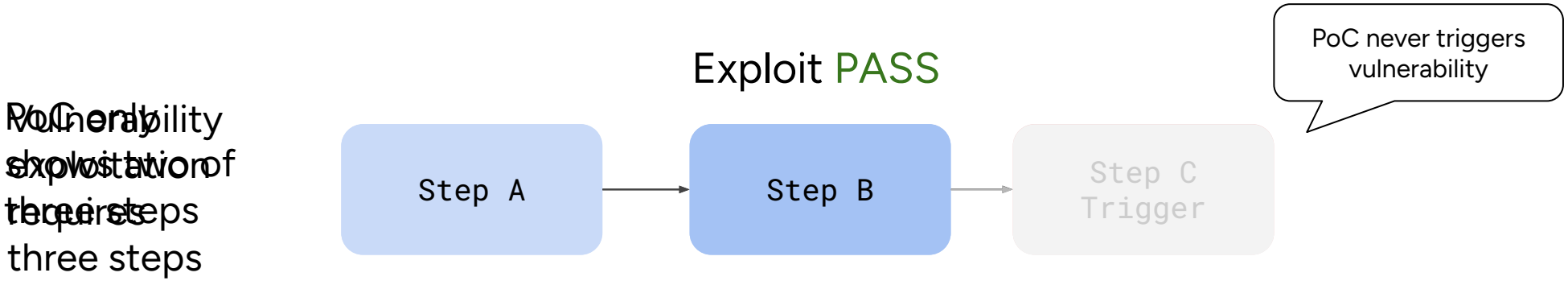
Manual analysis confirms patch failures reflect the intended exploit, not arbitrary breakage (on a subset)

Discussion

Failure mode: Incomplete Reach

Issue: PoC reaches patched code, but does not demonstrate the security violation.

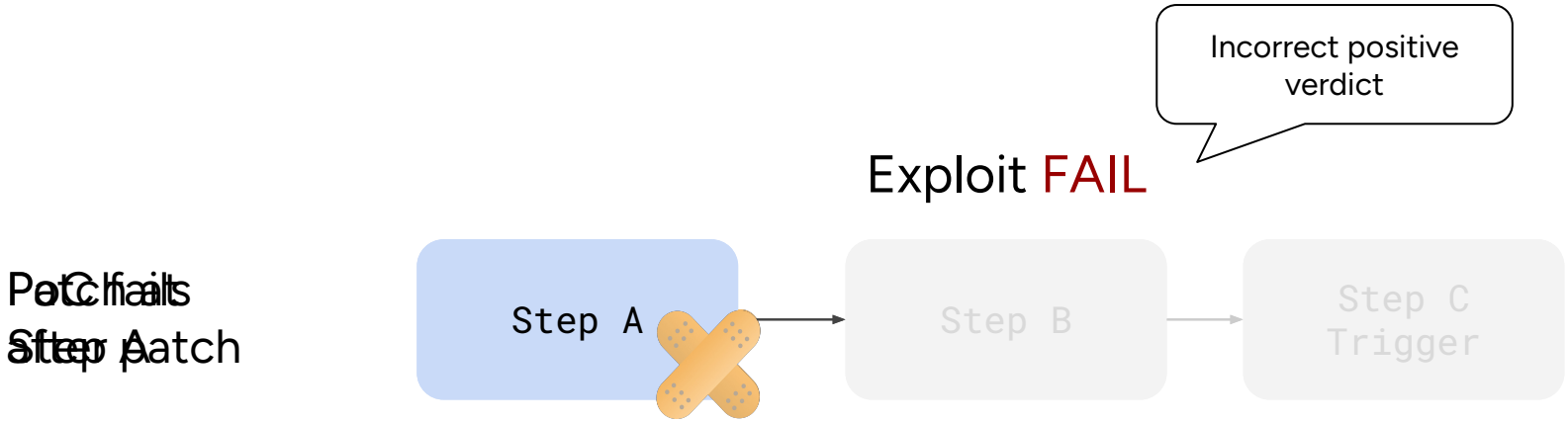
Risk: False positive–oracle marks as correct



Failure mode: Incomplete Reach

Issue: PoC reaches patched code, but does not demonstrate the security violation.

Risk: False positive–oracle marks as correct



This is a real false-positive risk. Still, with minimal patches, we hypothesize reaching patched code is meaningful evidence.

Alternative oracle designs

Oracle choice determines what “correct” means.

	Signal type	Online?	Strength	Limitation
Task-specific (PoCGen)	Observable symptom	Yes	Guides generation	May pass for wrong reason
Profitability (A1)	Net balance change	Yes	Hard to fake	Misses non-profitable vuln
Assertion-based (REX)	Foundry assertion	No	Easy to automate	May not align with the vuln
Patch-based (PoCo) (EVMBench*)	Behavioral diff	No	Encodes causal change	Requires minimal trusted patch

*reverse: PoC validates patch. See also “Do automated fixes truly mitigate smart contract exploits?” Bobadilla, Jin, Monperrus, TSE 2025.

Patch-based oracles give up online guidance, but works well across vulnerability types

Simsek, Deniz, Aryaz Eghbali, and Michael Pradel. "Pocgen: Generating proof-of-concept exploits for vulnerabilities in npm packages." arXiv:2506.04962 (2025),
Gervais, Arthur, and Liyi Zhou. "Ai agent smart contract exploit generation." *arXiv:2507.05558* (2025), Xiao, Zeke, et al.
"Prompt to pwn: Automated exploit generation for smart contracts." arXiv:2508.01371 (2025), Wang, Justin, et al. "EVMbench: Evaluating AI Agents on Smart Contract Security." arXiv:2603.04915 (2026)

Conclusion

Summary

Takeaway: High-quality security patches provide a **strong oracle for PoC correctness** in the smart contract domain

Contribution:

- PoCo: an agentic blueprint for smart contract PoC generation
- Proof-of-Patch: a benchmark and patch-based evaluation methodology for logical correctness

PoCo: Agentic Exploit Generation for Smart Contracts

Vivi Andersson, Sofia Bobadilla*, Harald Hobbelhagen, Martin Monperrus, 2025.*

ACM TOSEM Special issue on Agentic AI in Software.

Thanks to
Sofia Bobadilla