PoCo: Agentic PoC Exploit Generation for Smart Contracts

Vivi Andersson* <vivia@kth.se>
With Sofia Bobadilla*, Harald Hobbelhagen, and Martin Monperrus

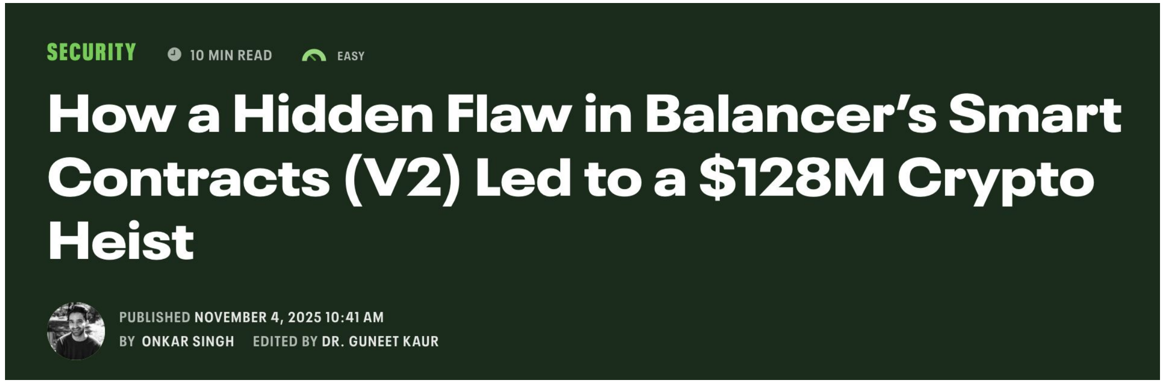
CDIS Retreat 2025-11-05

Smart contracts are consistently hacked, leading to loss of finances and trust

- **Open Code, Open Targets**
  → Adversaries can inspect and identify vulnerabilities

- **Irreversible Actions**
  → Exploits are permanent; no rollback

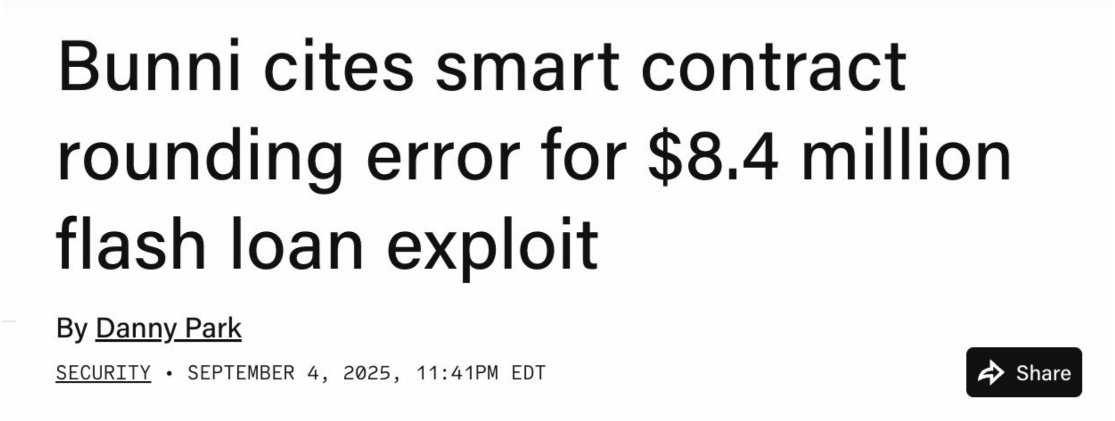
- **High Stakes**
  → Secures critical infrastructure and billions in assets



SECURITY 10 MIN READ EASY

How a Hidden Flaw in Balancer's Smart Contracts (V2) Led to a $128M Crypto Heist

PUBLISHED NOVEMBER 4, 2025 10:41 AM
BY ONKAR SINGH EDITED BY DR. GUNEET KAUR



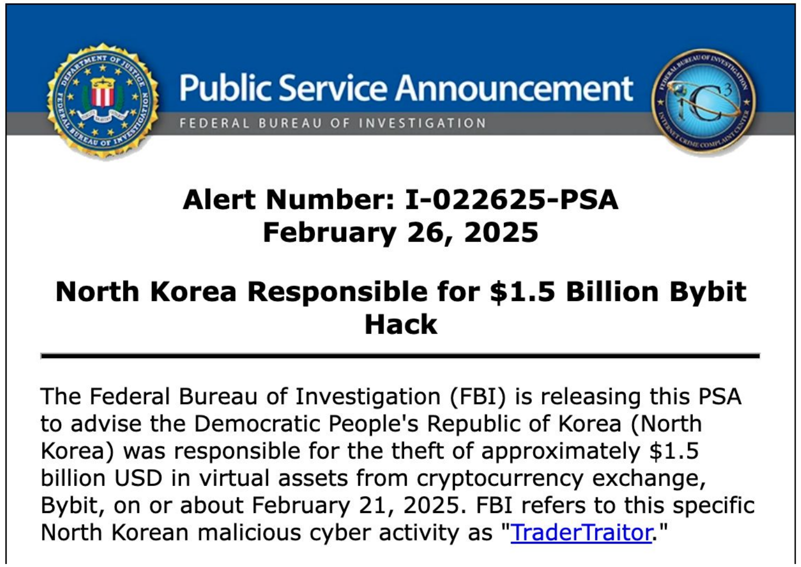
Bunni cites smart contract rounding error for $8.4 million flash loan exploit

By Danny Park

SECURITY • SEPTEMBER 4, 2025, 11:41PM EDT

DPRK’s Blockchain Cyber Operations

- DPRK systematically targets smart contracts as part of cyber operations [1]
- 2024: $1.34 billion stolen [2]
- 2025: > $2B [1] of which $1.5 billion in single Bybit hack
- UN reporting: Proceeds fund WMD & ballistic missile programs [1, 3]



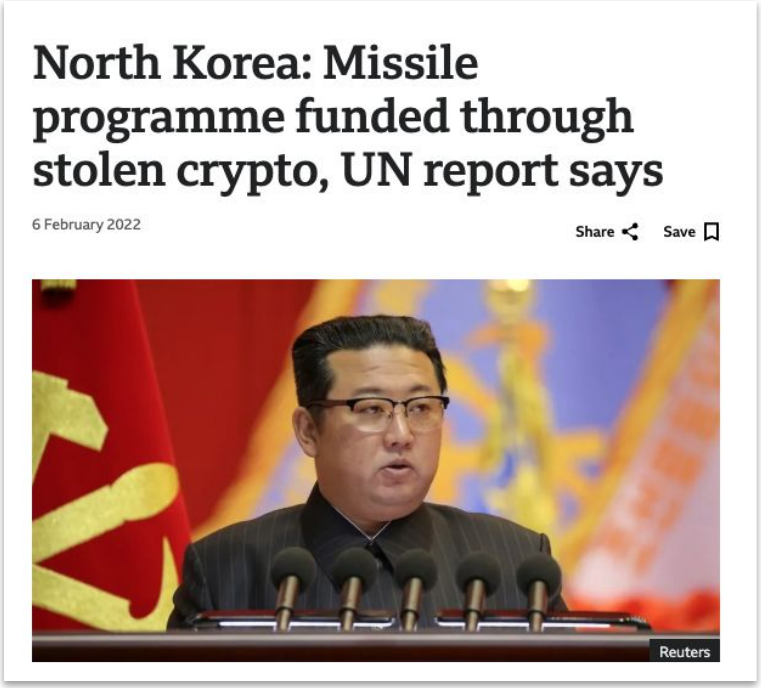
Public Service Announcement

FEDERAL BUREAU OF INVESTIGATION

Alert Number: I-022625-PSA
February 26, 2025

North Korea Responsible for $1.5 Billion Bybit Hack

The Federal Bureau of Investigation (FBI) is releasing this PSA to advise the Democratic People's Republic of Korea (North Korea) was responsible for the theft of approximately $1.5 billion USD in virtual assets from cryptocurrency exchange, Bybit, on or about February 21, 2025. FBI refers to this specific North Korean malicious cyber activity as "TraderTraitor."



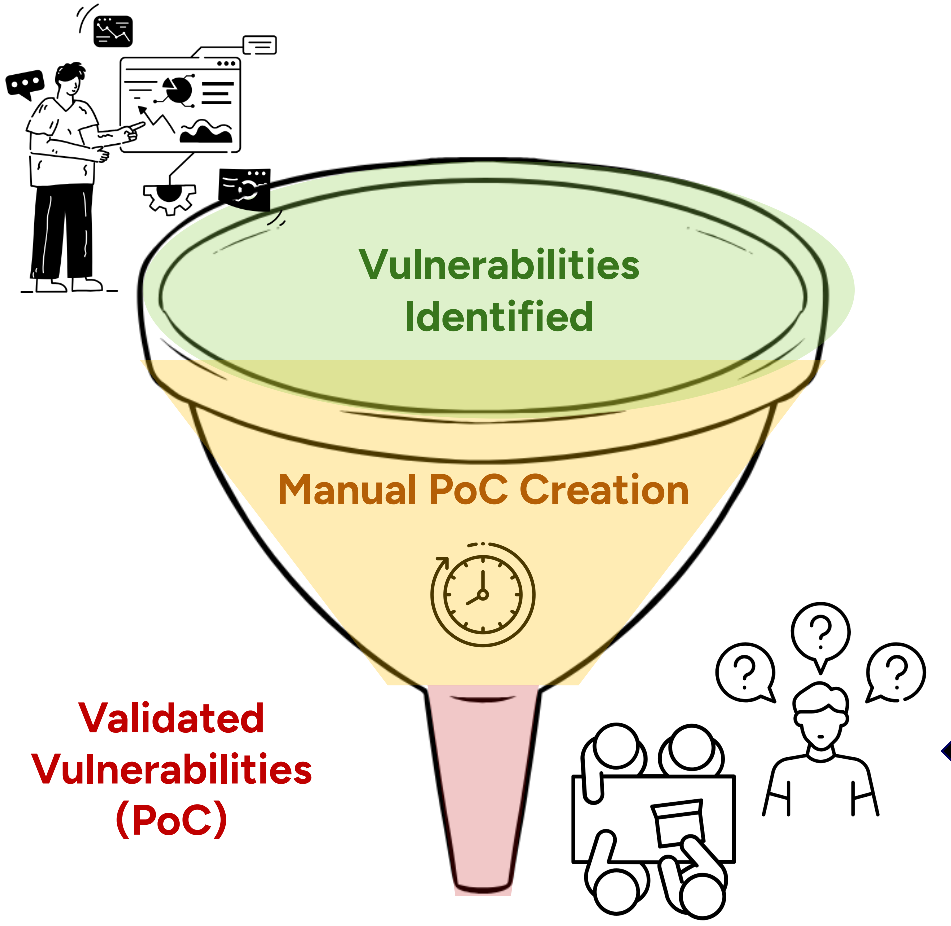
North Korea: Missile programme funded through stolen crypto, UN report says

6 February 2022

[1] Multilateral Sanctions Monitoring Team (MSMT). The DPRK’s Violation and Evasion of UN Sanctions through Cyber and Information Technology Worker Activities. MSMT, 2025,
[2] Chainalysis. "Crypto Hacking: $2.2 Billion Stolen in 2024 but Hacked Volumes Slow in Second Half." Chainalysis, 2025, https://www.chainalysis.com/blog/crypto-hacking-stolen-funds-2025/

In smart contract audits, PoCs are essential: they validate vulnerabilities, demonstrate exploitability, and drive patch prioritization.

Audits depend on human expertise, but auditors prioritize discovery over creating PoCs, a process that is slow and expertise-heavy.



Vulnerabilities Identified

Manual PoC Creation

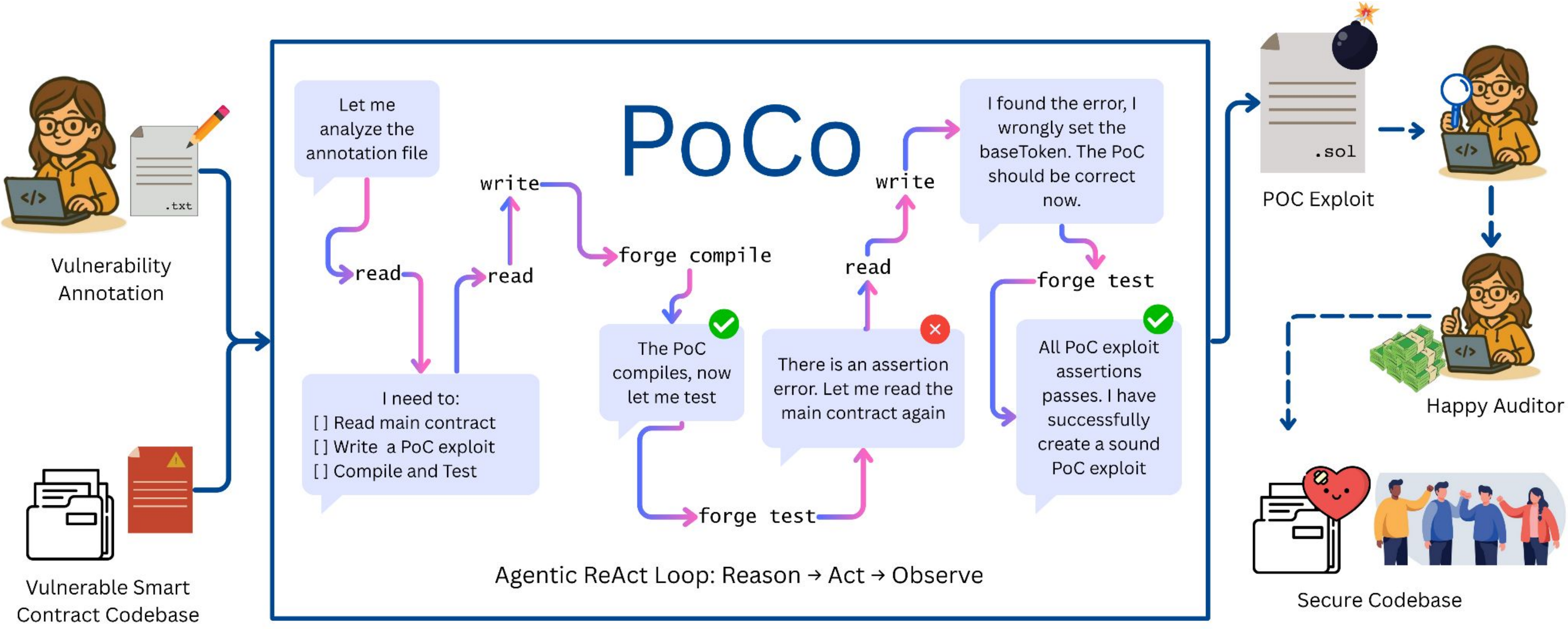
Validated Vulnerabilities (PoC)

Without a PoC, vulnerability reports remain unvalidated, making their true impact difficult for protocol developers to assess.

PoCo

Agentic Smart Contract PoC Exploit Generation

PoCo generates executable vulnerability PoCs from human-written vulnerability descriptions



Vulnerability Annotation

Vulnerable Smart Contract Codebase

PoCo

Let me analyze the annotation file

read

I need to:
[ ] Read main contract
[ ] Write a PoC exploit
[ ] Compile and Test

read

write

forge compile

The PoC compiles, now let me test

forge test

There is an assertion error. Let me read the main contract again

read

write

I found the error, I wrongly set the baseToken. The PoC should be correct now.

forge test

All PoC exploit assertions passes. I have successfully create a sound PoC exploit

Agentic ReAct Loop: Reason → Act → Observe

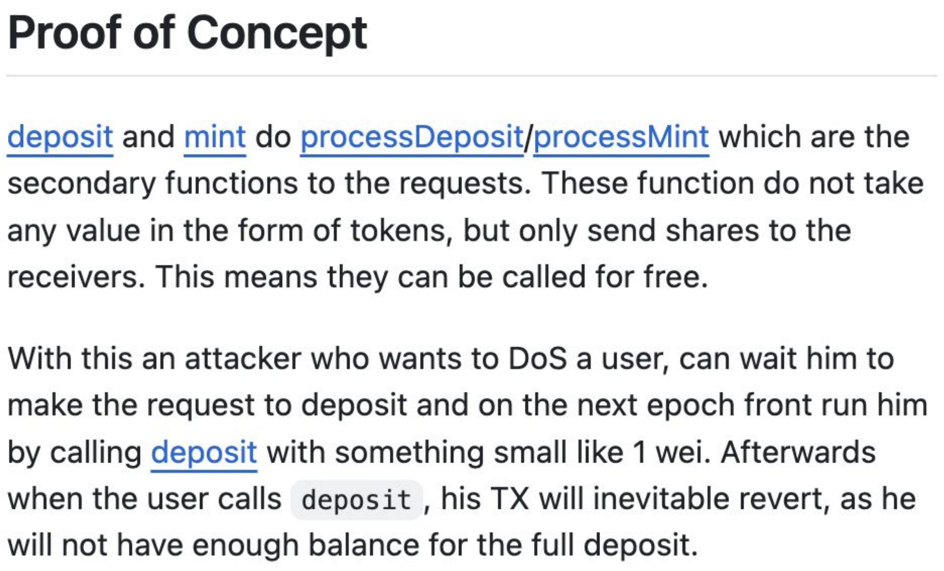
POC Exploit

Happy Auditor

Secure Codebase

Smart Contract PoC Exploit

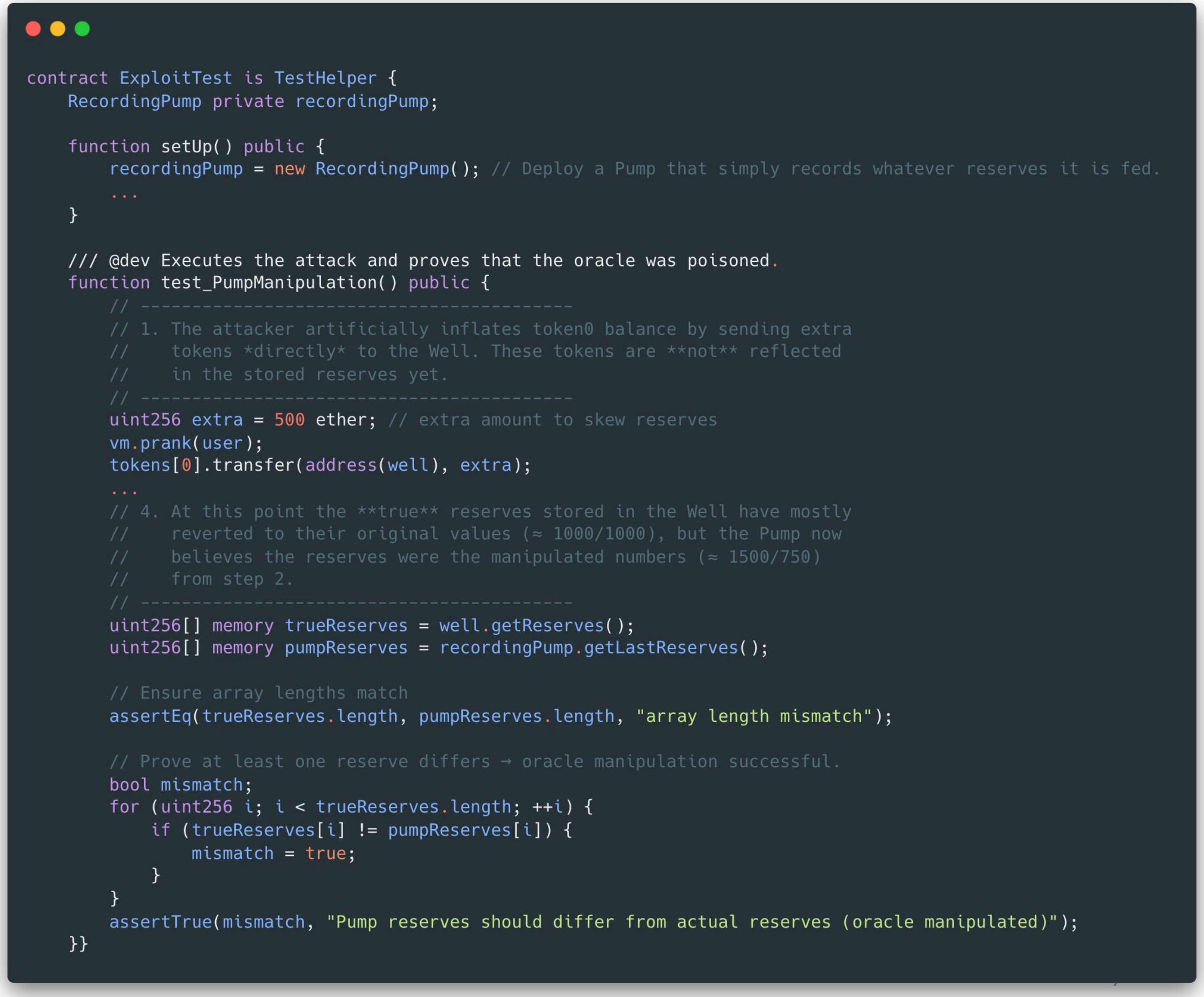
Executable PoCs = verifiable vulnerability findings



Proof of Concept

deposit and mint do processDeposit/processMint which are the secondary functions to the requests. These function do not take any value in the form of tokens, but only send shares to the receivers. This means they can be called for free.

With this an attacker who wants to DoS a user, can wait him to make the request to deposit and on the next epoch front run him by calling deposit with something small like 1 wei. Afterwards when the user calls `deposit`, his TX will inevitable revert, as he will not have enough balance for the full deposit.

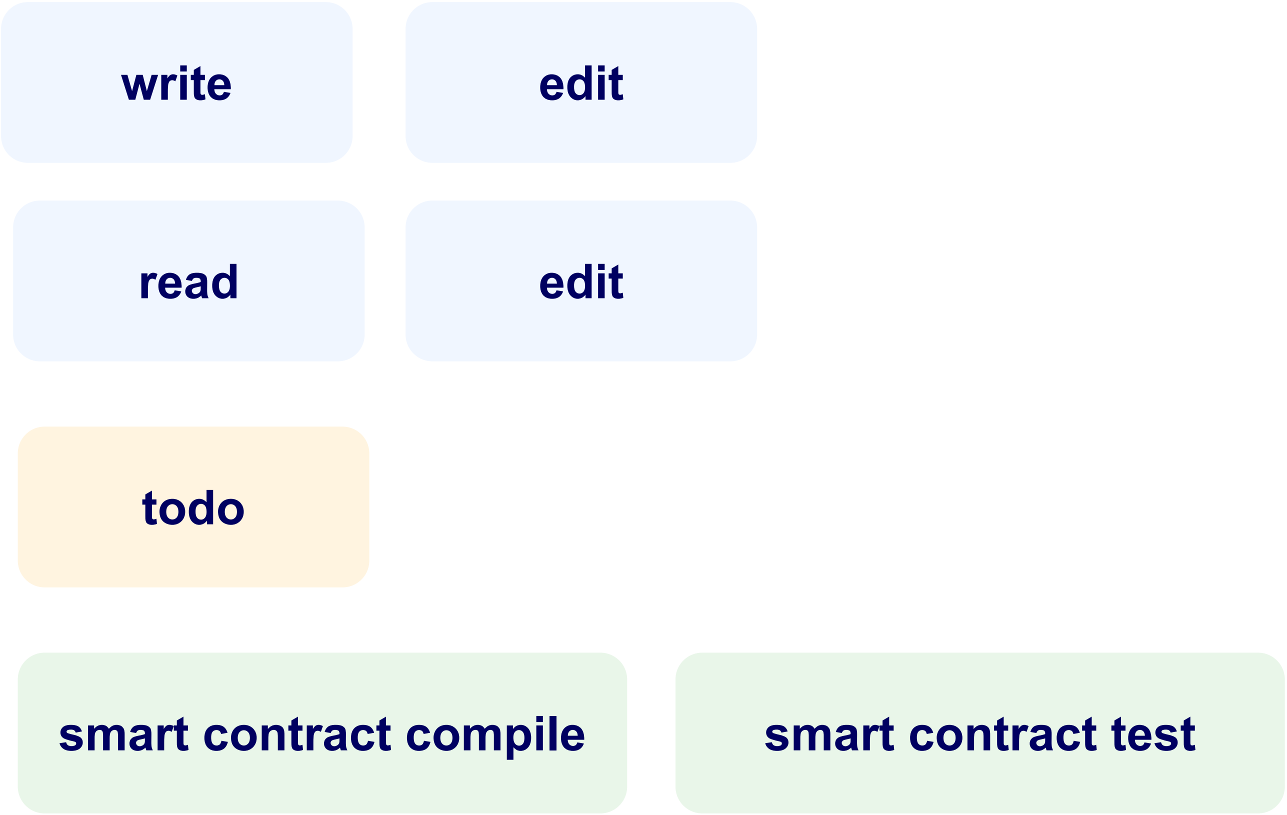


```
contract ExploitTest is TestHelper {
    RecordingPump private recordingPump;

    function setUp() public {
        recordingPump = new RecordingPump(); // Deploy a Pump that simply records whatever reserves it is fed.
        ...
    }

    /// @dev Executes the attack and proves that the oracle was poisoned.
    function test_PumpManipulation() public {
        // ------------------------------------------
        // 1. The attacker artificially inflates token0 balance by sending extra
        //    tokens *directly* to the Well. These tokens are **not** reflected
        //    in the stored reserves yet.
        // ------------------------------------------
        uint256 extra = 500 ether; // extra amount to skew reserves
        vm.prank(user);
        tokens[0].transfer(address(well), extra);
        ...
        // 4. At this point the **true** reserves stored in the Well have mostly
        //    reverted to their original values (≈ 1000/1000), but the Pump now
        //    believes the reserves were the manipulated numbers (≈ 1500/750)
        //    from step 2.
        // ------------------------------------------
        uint256[] memory trueReserves = well.getReserves();
        uint256[] memory pumpReserves = recordingPump.getLastReserves();

        // Ensure array lengths match
        assertEq(trueReserves.length, pumpReserves.length, "array length mismatch");

        // Prove at least one reserve differs → oracle manipulation successful.
        bool mismatch;
        for (uint256 i; i < trueReserves.length; ++i) {
            if (trueReserves[i] != pumpReserves[i]) {
                mismatch = true;
            }
        }
        assertTrue(mismatch, "Pump reserves should differ from actual reserves (oracle manipulated)");
}}
```

PoCo: Architecture

LLM augmented with tools, acting in ReAct loop

- Basic tools: Codebase exploration
- Planning tool: task breakdown
- Smart contract tools: task goal feedback
    - Executed in sandboxed env



write

edit

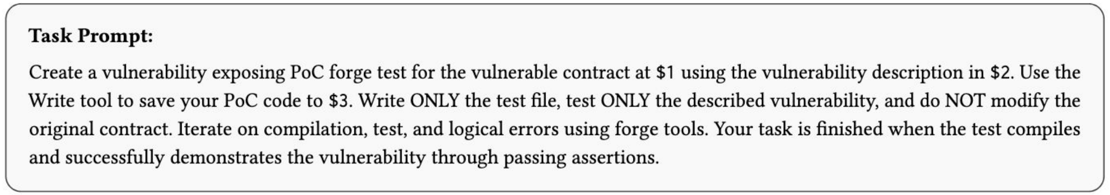
read

edit

todo

smart contract compile

smart contract test



Task Prompt:

Create a vulnerability exposing PoC forge test for the vulnerable contract at $1 using the vulnerability description in $2. Use the Write tool to save your PoC code to $3. Write ONLY the test file, test ONLY the described vulnerability, and do NOT modify the original contract. Iterate on compilation, test, and logical errors using forge tools. Your task is finished when the test compiles and successfully demonstrates the vulnerability through passing assertions.

Demo!

Evaluation

Research Questions

RQs

1. Can PoCo generate well-formed PoC exploits?

   We asses whether the PoC compiles and that the assertions pass

2. Can PoCo generate logically correct PoC exploits?

   We report whether the exploits are mitigated by the ground-truth patch

3. How do annotation detail affect the results?

   We vary the level of detail on the annotations and report their logical correctness

Testing PoCo on Real-World Vulnerabilities

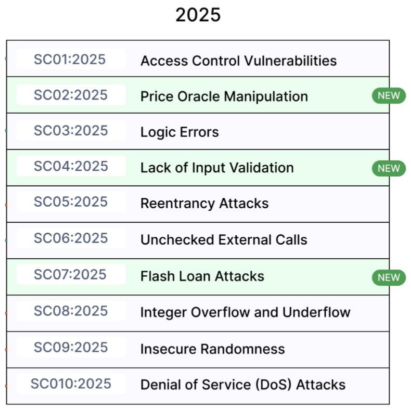
Proof-of-Patch

- 23 real-world vulnerabilities from manually verified security audit reports
- Patches: developer-accepted ground truth

Baselines

- Single-pass prompting
- Workflow prompting (iterative loop)

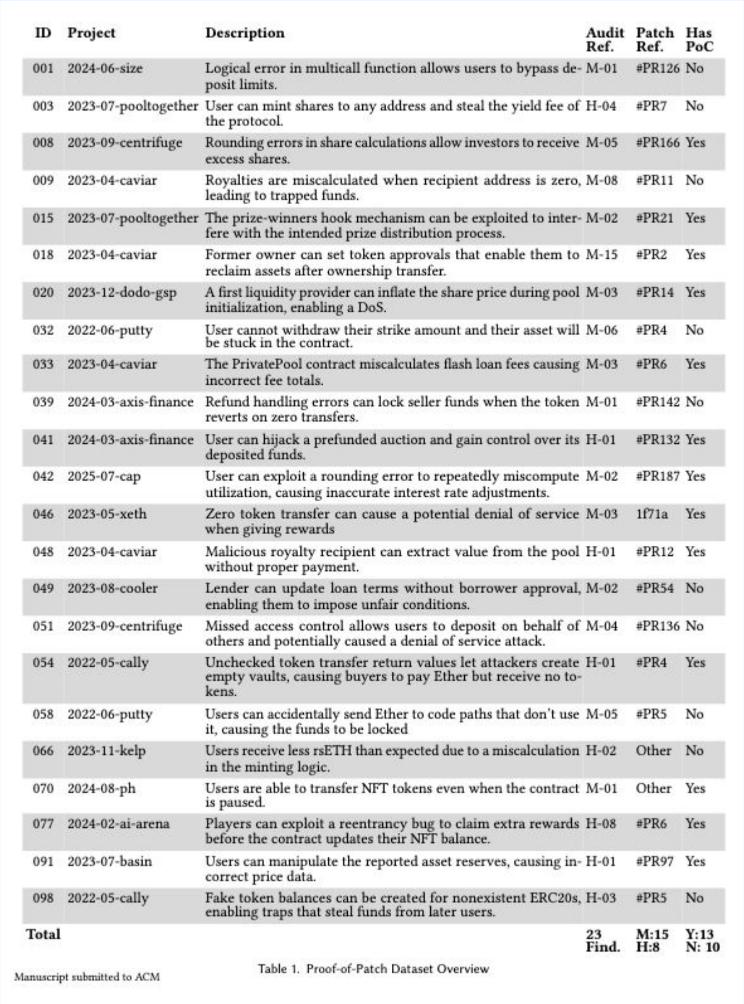
- Models: Claude Sonnet 4.5, GLM 4.6, and o3
- Limit generation: $3 USD or 10 tool calls



2025

SC01:2025	Access Control Vulnerabilities	
SC02:2025	Price Oracle Manipulation	NEW
SC03:2025	Logic Errors	
SC04:2025	Lack of Input Validation	NEW
SC05:2025	Reentrancy Attacks	
SC06:2025	Unchecked External Calls	
SC07:2025	Flash Loan Attacks	NEW
SC08:2025	Integer Overflow and Underflow	
SC09:2025	Insecure Randomness	
SC010:2025	Denial of Service (DoS) Attacks	

Dataset: **Proof-of-Patch**



ID	Project	Description	Audit Ref.	Patch Ref.	Has PoC
001	2024-06-size	Logical error in multicall function allows users to bypass deposit limits.	M-01	#PR126	No
003	2023-07-pooltogether	User can mint shares to any address and steal the yield fee of the protocol.	H-04	#PR7	No
008	2023-09-centrifuge	Rounding errors in share calculations allow investors to receive excess shares.	M-05	#PR166	Yes
009	2023-04-caviar	Royalties are miscalculated when recipient address is zero, leading to trapped funds.	M-08	#PR11	No
015	2023-07-pooltogether	The prize-winners hook mechanism can be exploited to interfere with the intended prize distribution process.	M-02	#PR21	Yes
018	2023-04-caviar	Former owner can set token approvals that enable them to reclaim assets after ownership transfer.	M-15	#PR2	Yes
020	2023-12-dodo-gsp	A first liquidity provider can inflate the share price during pool initialization, enabling a DoS.	M-03	#PR14	Yes
032	2022-06-putty	User cannot withdraw their strike amount and their asset will be stuck in the contract.	M-06	#PR4	No
033	2023-04-caviar	The PrivatePool contract miscalculates flash loan fees causing incorrect fee totals.	M-03	#PR6	Yes
039	2024-03-axis-finance	Refund handling errors can lock seller funds when the token reverts on zero transfers.	M-01	#PR142	No
041	2024-03-axis-finance	User can hijack a prefunded auction and gain control over its deposited funds.	H-01	#PR132	Yes
042	2025-07-cap	User can exploit a rounding error to repeatedly miscompute utilization, causing inaccurate interest rate adjustments.	M-02	#PR187	Yes
046	2023-05-xeth	Zero token transfer can cause a potential denial of service when giving rewards	M-03	1f71a	Yes
048	2023-04-caviar	Malicious royalty recipient can extract value from the pool without proper payment.	H-01	#PR12	Yes
049	2023-08-cooler	Lender can update loan terms without borrower approval, enabling them to impose unfair conditions.	M-02	#PR54	No
051	2023-09-centrifuge	Missed access control allows users to deposit on behalf of others and potentially caused a denial of service attack.	M-04	#PR136	No
054	2022-05-cally	Unchecked token transfer return values let attackers create empty vaults, causing buyers to pay Ether but receive no tokens.	H-01	#PR4	Yes
058	2022-06-putty	Users can accidentally send Ether to code paths that don't use it, causing the funds to be locked	M-05	#PR5	No
066	2023-11-kelp	Users receive less rsETH than expected due to a miscalculation in the minting logic.	H-02	Other	No
070	2024-08-ph	Users are able to transfer NFT tokens even when the contract is paused.	M-01	Other	Yes
077	2024-02-ai-arena	Players can exploit a reentrancy bug to claim extra rewards before the contract updates their NFT balance.	H-08	#PR6	Yes
091	2023-07-basin	Users can manipulate the reported asset reserves, causing incorrect price data.	H-01	#PR97	Yes
098	2022-05-cally	Fake token balances can be created for nonexistent ERC20s, enabling traps that steal funds from later users.	H-03	#PR5	No
Total			**23 Find.**	**M:15 H:8**	**Y:13 N: 10**

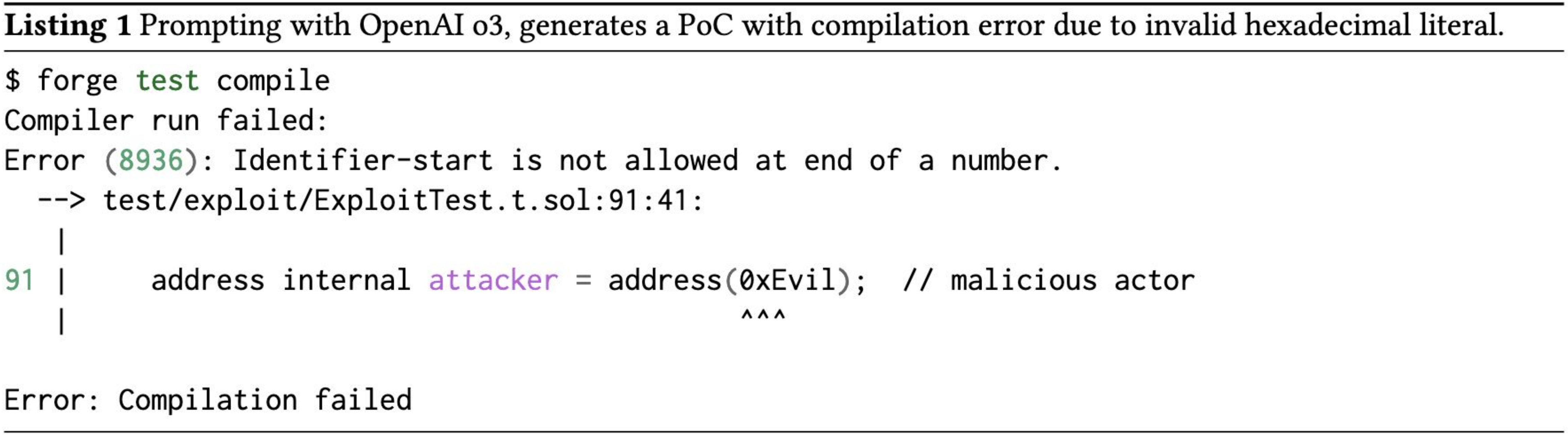
Table 1. Proof-of-Patch Dataset Overview

Manuscript submitted to ACM

PoCo Generates Well-formed & Logically Correct PoCs

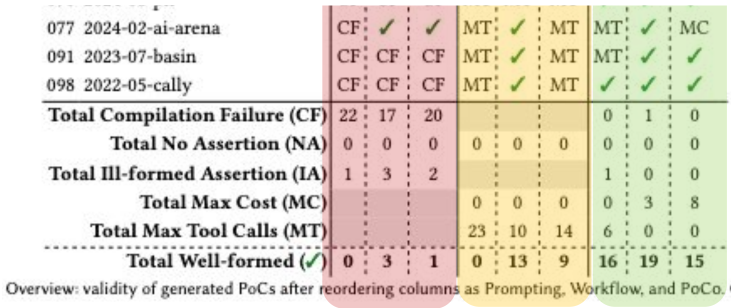
RQ1: Well-formed PoCs

RQ2: Logically correct PoCs



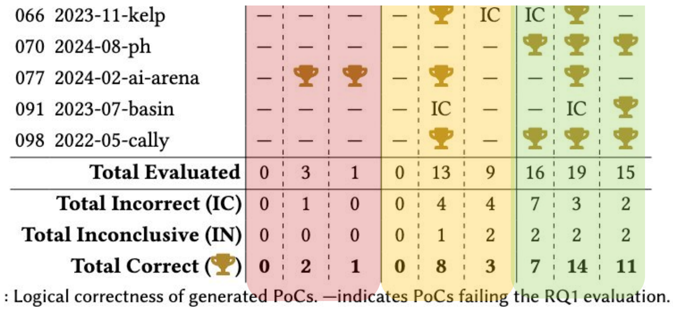
Listing 1 Prompting with OpenAI o3, generates a PoC with compilation error due to invalid hexadecimal literal.

```
$ forge test compile
Compiler run failed:
Error (8936): Identifier-start is not allowed at end of a number.
  --> test/exploit/ExploitTest.t.sol:91:41:
   |
91 |     address internal attacker = address(0xEvil);  // malicious actor
   |                                         ^^^

Error: Compilation failed
```



077 2024-02-ai-arena	CF	✓	✓	MT	✓	MT	MT	✓	MC
091 2023-07-basin	CF	CF	CF	MT	✓	MT	MT	✓	✓
098 2022-05-cally	CF	CF	CF	MT	✓	MT	✓	✓	✓
Total Compilation Failure (CF)	22	17	20				0	1	0
Total No Assertion (NA)	0	0	0	0	0	0	0	0	0
Total Ill-formed Assertion (IA)	1	3	2				1	0	0
Total Max Cost (MC)				0	0	0	0	3	8
Total Max Tool Calls (MT)				23	10	14	6	0	0
Total Well-formed (✓)	**0**	**3**	**1**	**0**	**13**	**9**	**16**	**19**	**15**

Overview: validity of generated PoCs after reordering columns as Prompting, Workflow, and PoCo.



066 2023-11-kelp	—	—	—	—	🏆	IC	IC	🏆	—
070 2024-08-ph	—	—	—	—	—	—	🏆	🏆	🏆
077 2024-02-ai-arena	—	🏆	🏆	—	🏆	—	—	🏆	—
091 2023-07-basin	—	—	—	—	IC	—	—	IC	🏆
098 2022-05-cally	—	—	—	—	🏆	—	🏆	🏆	🏆
Total Evaluated	0	3	1	0	13	9	16	19	15
Total Incorrect (IC)	0	1	0	0	4	4	7	3	2
Total Inconclusive (IN)	0	0	0	0	1	2	2	2	2
Total Correct (🏆)	**0**	**2**	**1**	**0**	**8**	**3**	**7**	**14**	**11**

: Logical correctness of generated PoCs. —indicates PoCs failing the RQ1 evaluation.

Trajectory: Price Oracle Manipulation in Basin DEX

1. Reconnaissance (rounds 1- 25)

PoCo: I’ll analyze the vulnerability and create a PoC test. Let me start by examining the vulnerable contract and the vulnerability description.

2. Initial Exploit Writing (round 26)

‘forge test‘ aborts with Error7576 — ‘console.log‘ used without importing ‘forge-std/Test.sol‘.

3. Compilation Debugging (rounds 27-46)

4. Test Assertion Refinement (rounds 47– 58)

PoCo: I see, the IPump interface only has update, not read. Let me fix the TrackingPump. Tool (seq. 49): Test suite run: two passes,

[2PASS 1 FAIL]

test_exploit_sync_oracle_manipulation still fails because the oracle assertion expects post-swap reserves. The issue is in line151 — the pump receives reserves before the swap adds to them. I’ll adjust the assertion to compare with reservesAfterSync

[3 PASS]

Takeaways

1) **PoCo automates the PoC creation bottleneck:** transforming vulnerability descriptions into executable exploits

2) **Validates vulnerability reports:** gives protocol developers concrete evidence to assess true impact and prioritize patches

3) **Outperforms traditional ML baselines**