Introduction to DeFi

10 / 4

CONTENT

Common Attack Vector

Smart Contract Security

- This course focuses on solidity security, It’s important to remember that it’s not overly complex.
- Every DeFi protocol engineer strives to protect their project, and the responsibility for securing smart contracts doesn’t fall solely on the auditors.
- What we will cover today: (1) access control, (2) arbitrary call, (3) default visibility, (4) reentrancy issue, (5) bad randomness, (6) ecrecover issue, (7) signature replay.

Access Control

Case 1: Do not have `onlyOwner` modifier, allowing attack to set important parameter.

Protocol: GFOX

Problematic Function: `setMerkleRoot` function

- No `onlyOwner` modifier

Impact:

- Anyone can reset the Merkle root to include themself in the whitelist

Transaction details, we will discuss more about this case & merkle tree later in the course.

Access Control

Real-World Case Study: I accidentally kill it.

Problem: The owner does not init the wallet

(1) The initWallet is not invoke when creation

(2) There is arbitrary delegatecall issue in the contract.

(3) The user call the kill function that contains `kill`

(4) Not an attack incident, an user is just learning smart contract - A $280M lesson learned

Analysis Contract

Access Control

Case 2: Do not have `initializer` modifier, allowing attack to re-initialize the contract

Protocol: 88mph

Problematic Function: `init` function

- No `initializer` modifier for initialization
- No `onlyOwner` verification

Impact:

- Anyone can be the owner
- Able to reset system token

Access Control

Case 3: Do not verify the caller in flash loan receive function

Problematic Function: `executeOperation` function (Aave flash loan receiver function)

Uniswap: uniswapV2Call
Balancer: receiveFlashLoan
Aave: executeOperation

- Not verify `initiator` address - anyone can initiate flash loan to trigger the function
- Not verify the `msg.sender` - anyone can call the `executeOperation` directly.

Access Control

MEV Bot Case Study: Lack of Contract Verification Won't Protect You from Attacks

20221108	**MEV_0ad8**	**Unlimited external call**	**$282k**
20221014	MEVBOTa47b	Insufficient validation	$241 k
20220928	MEVBOT - Badc0de	Arbitrary call via callFunction	$94,304
20220913	MevBot private tx	Incorrect access control	$140 K
20231112	MEVBot_0xa247	Access Control	$150K
20231107	MEVbot	Access Control	$2M
20231112	MEVBot_0x8c2d	Access Control	$365K

Link

Access Control

Case 4: unprotected callback in ERC-721 and ERC-1155

Problematic Function: `onERC721Received` function

- The callback might contains some critical logics and not protected.
- Anyone can directly invoke the callback

Instance: ERC-721, ERC-1155

Default Function Visibility

Case 1: If you do not specify the function visibility, The default visibility is `public`, not `internal`.

Real World Case: CoinstoreNFT

- No protected `burn` function
- Not verify the token owner

Real World Case: The Sandbox

(1) No protected `_burn` function

Arbitrary Call

Case 1: Arbitrary call with `call` - calldata injection.

Some case study

MixedSwapRouter::algebraSwapCallback

MinterProxyV2::swap

SpectraRouter::_dispatch

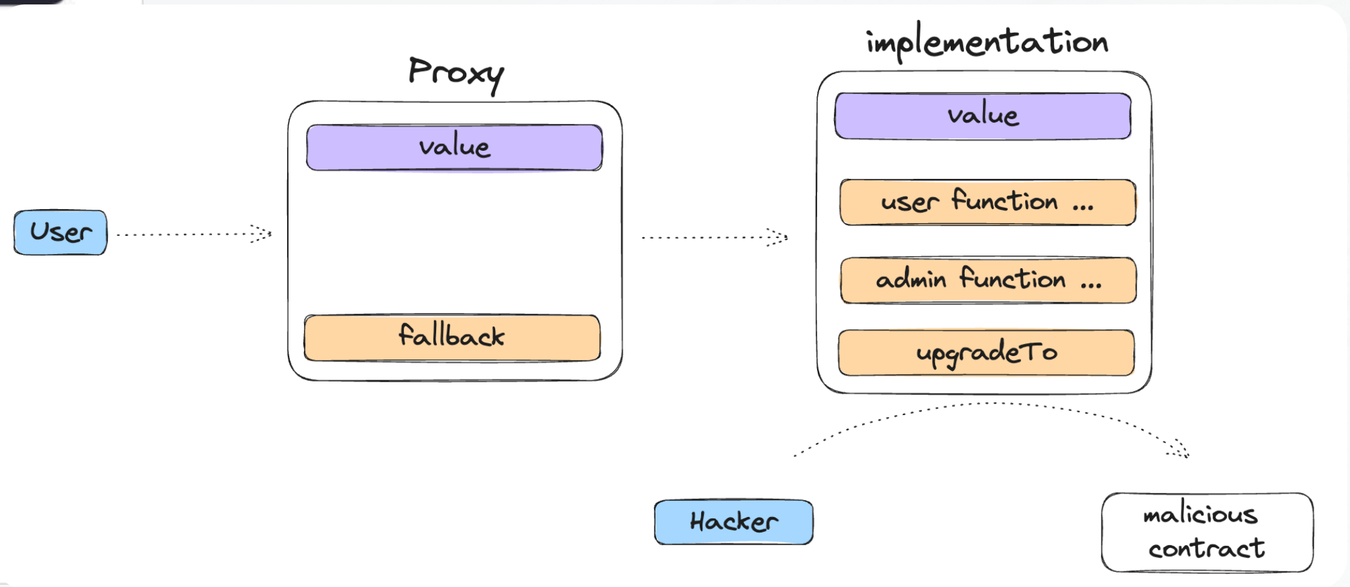
It a common attack vector. But it does not mean this pattern should never be used.

For example: DAO, Multisig Wallet

Arbitrary Call

Case 1: Arbitrary call with `delegatecall` - Logic contract contains malicious operation `selfdestruct` in implementation contract.

- For example, the UUPS vulnerability



Reentrancy Issue

What is reentrancy issue? Check your system programming slides.

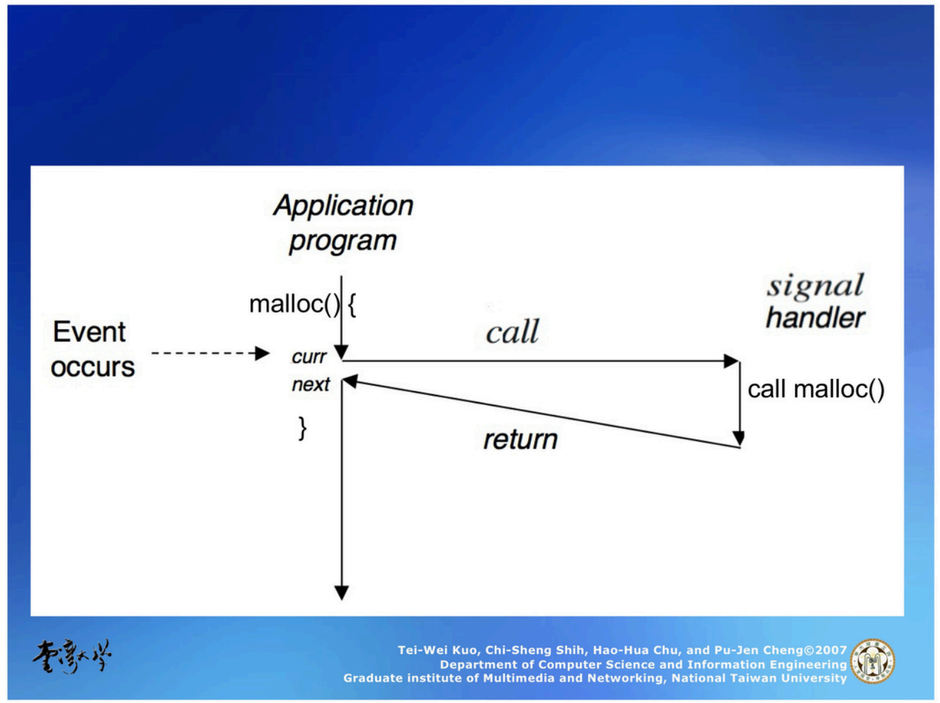
Reentrant Functions

- **When interrupts occur, a function could be called twice and causes problems.**
- **Potential Problem**
  - In the signal handler, we can't tell where the process was executing when the signal was caught!
- **A function is described as reentrant if it can be safely called recursively**

Tei-Wei Kuo, Chi-Sheng Shih, Hao-Hua Chu, and Pu-Jen Cheng©2007
Department of Computer Science and Information Engineering
Graduate institute of Multimedia and Networking, National Taiwan University

Reference: SP from PJ

An example in C: malloc



Reentrancy Issue

Different Types of Reentrancy

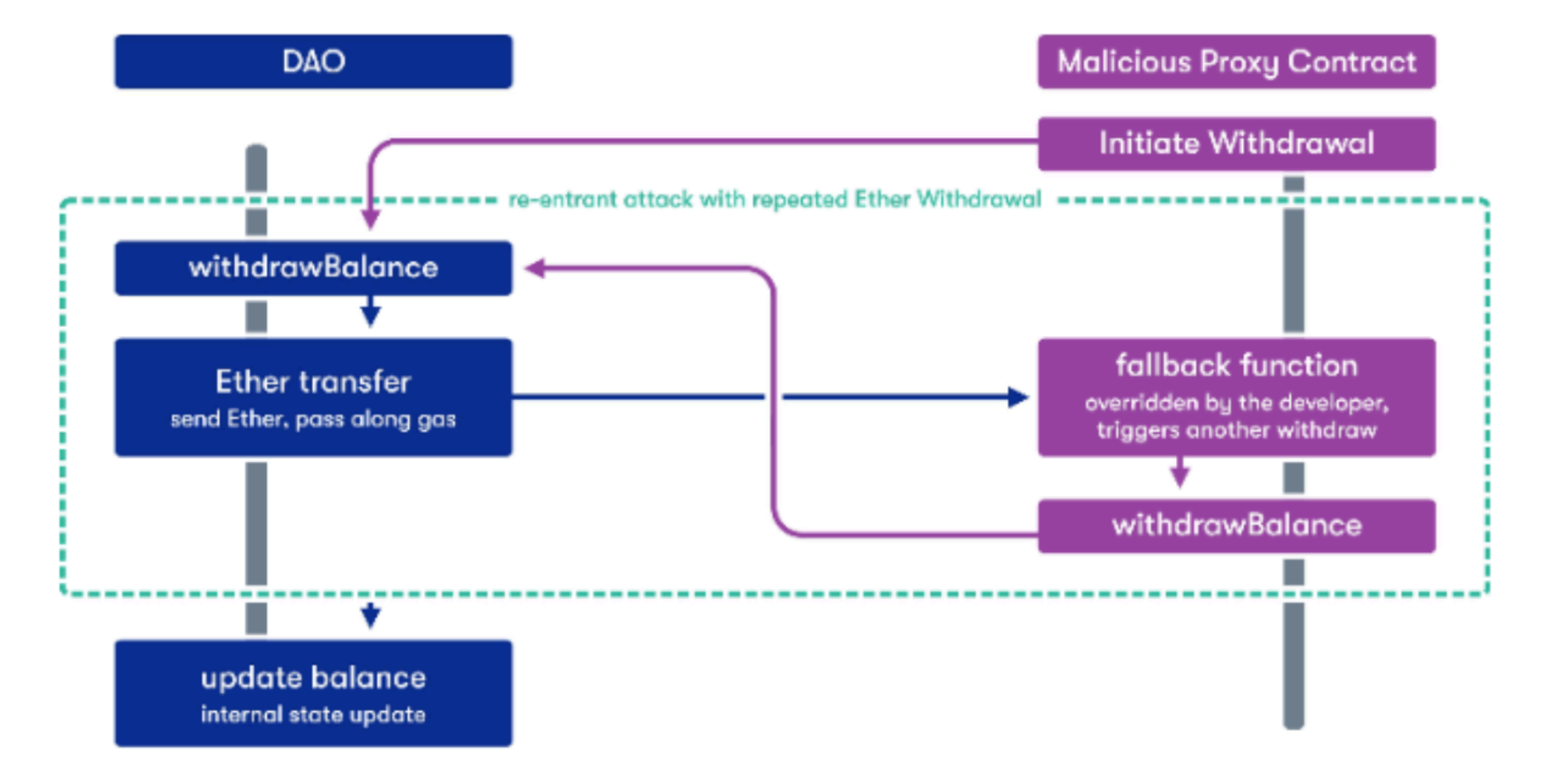
- Single function reentrancy
- Cross function reentrancy
- Cross contract reentrancy
- Cross chain reentrancy
- Read only reentrancy

The problem lies in the function that changes the state - not pure, view function

Note: the problem is in the `view` function

Reentrancy Issue

Case 1: Single function reentrancy



Example

Reentrancy Issue

Case 1: Cross function reentrancy

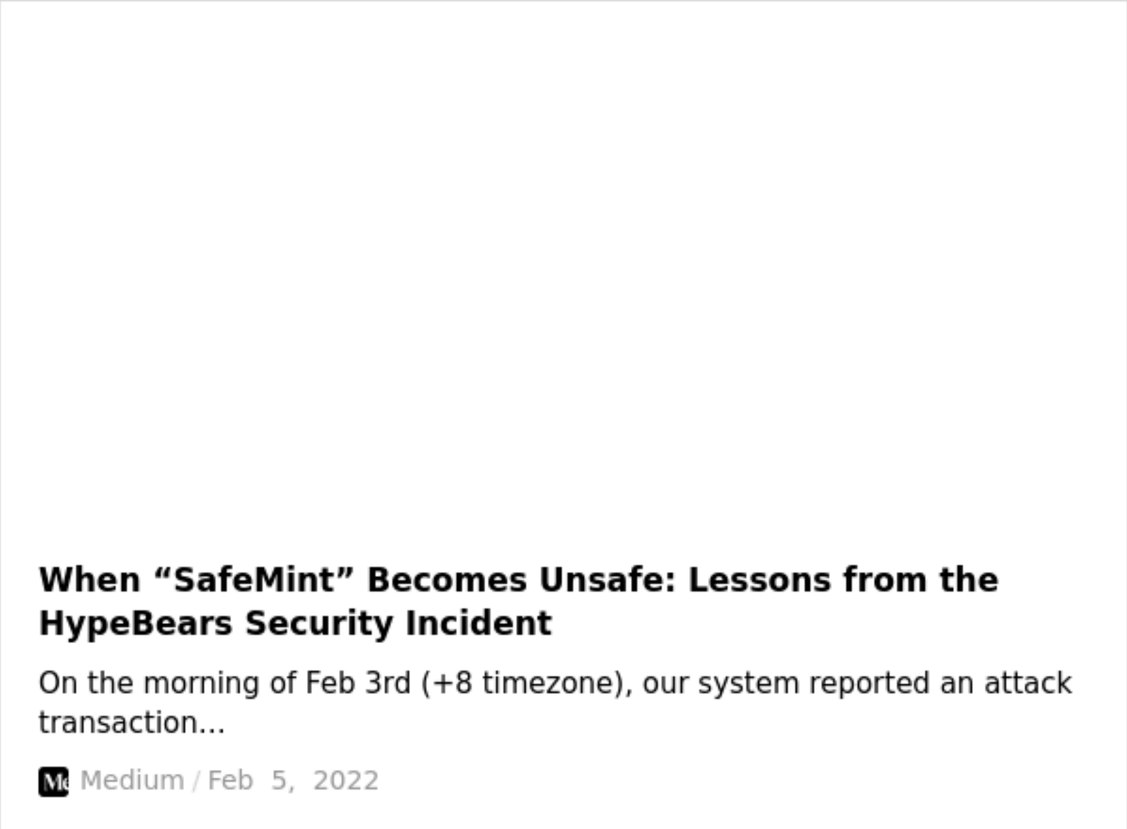
DFX Finance Hackâ€”Nov 10, 2022â€”Detailed Analysis

Overview On the 10th of November, 2022, DFX Finance, operating on an Ethereum-based protocol with the attack manifesting on the Polygon network, fell victim to a security breach involving a flash...

ImmuneBytes / Dec 1, 2023

Reentrancy Issue

Case 1: reentrancy issue in IERC721Receiver / IERC1155Receiver

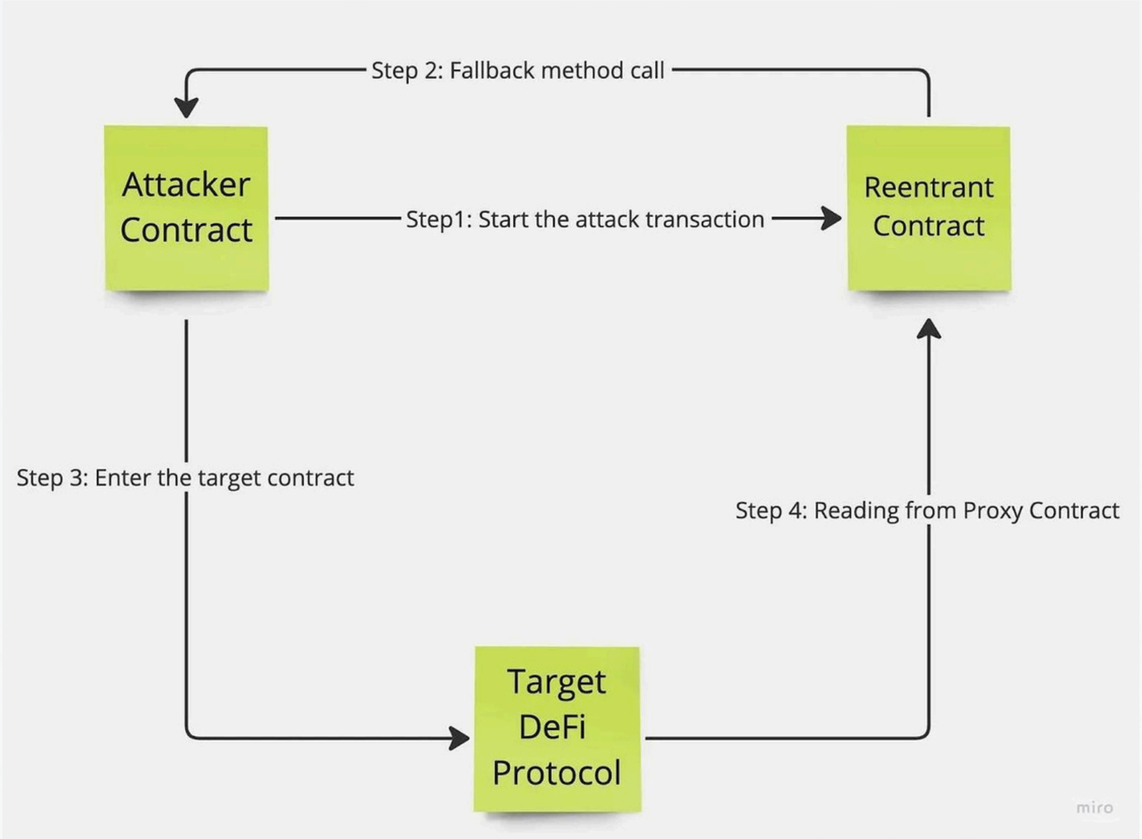


When `safeMint` become unsafe

Is ERC-20 safe? There are some variant:
ERC 223, 677, 777, and 1363

Reentrancy Issue

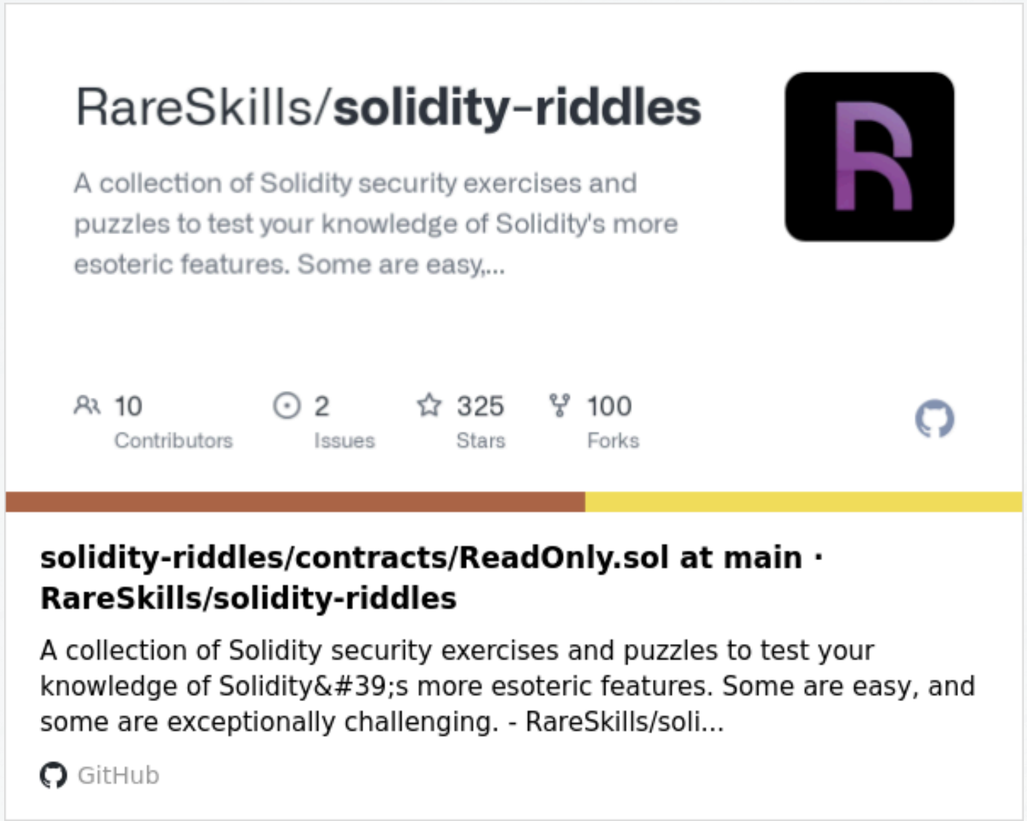
Case 1: Read Only Reentrancy



The problem is in the `view` function

Reentrancy Issue

Exercise: Read Only Reentrancy



Simple Example

Reentrancy Issue

Case 1: Curve LP Oracle Manipulation

What is an oracle, what is Curve, and what is an LP (liquidity provider)?

A Real-World Example

Reentrancy Issue

Protection 1: Use mutex lock

Non upgradeable version

Upgradeable version

Reentrancy Issue

Protection 2: Check-Effect-Interaction

- Check: validate all the input
- Effect: update the state variable
- Interaction: make external call

How to change our contract?

Bad Randomness

How do you get random number on the blockchain?

Bad Randomness in Solidity Smart Contracts

Introduction Bad randomness, often referred to as the "nothing is
attack, is a vulnerability in Solidity smart contracts deployed on th
Ethereum blockchain. This attack vector exploits the...

ImmuneBytes / Nov 1, 2023

SunWeb3Sec/
DeFiVulnLabs

To learn common smart contract vulnerabilities using Foundry!

13 Contributors · 1 Issue · 2k Stars · 289 Forks

DeFiVulnLabs/src/test/Randomness.sol at main · SunWeb3Sec/DeFiVulnLabs

To learn common smart contract vulnerabilities using Foundry! - SunWeb3Sec/DeFiVulnLabs

GitHub

Bad Randomness

Soluiton: Chainlink VRF

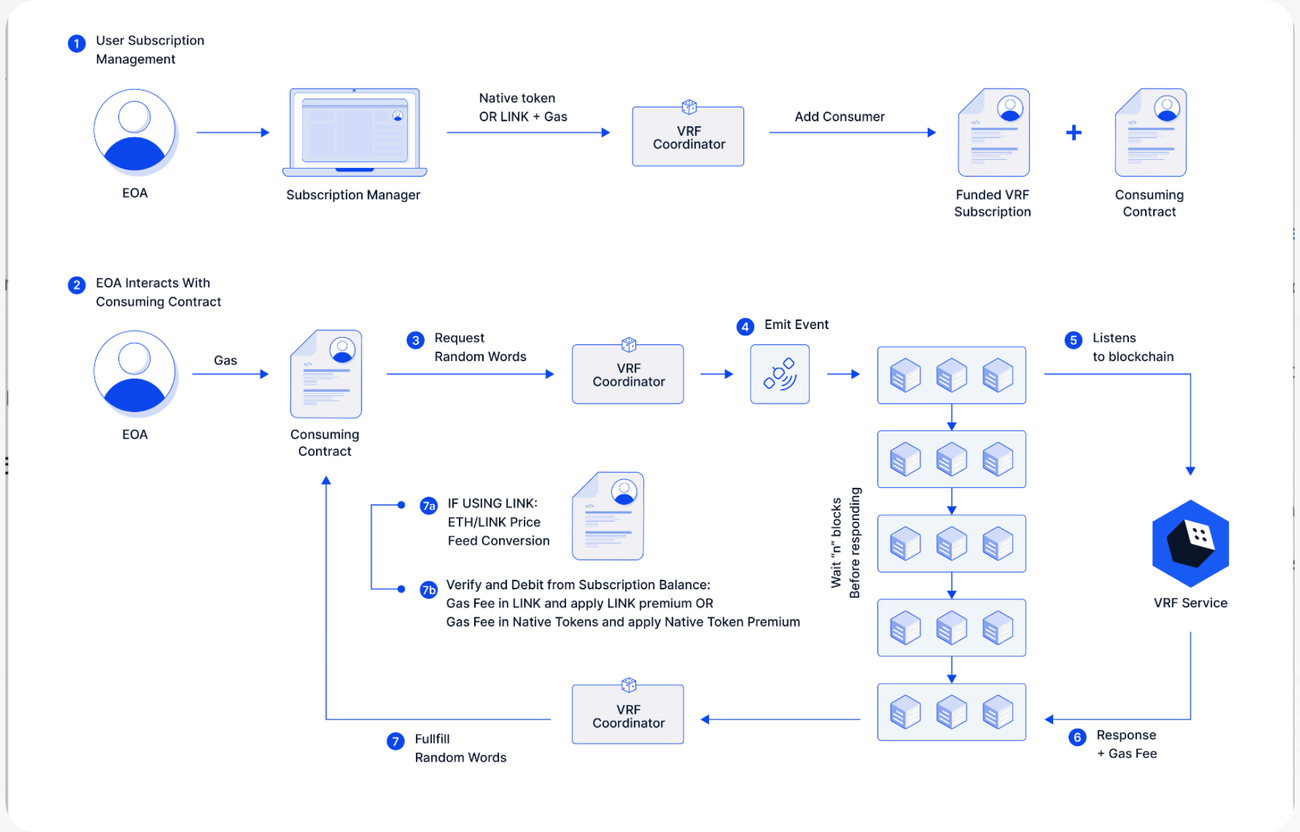
Generate Random Numbers for Smart Contracts using Chainlink VRF

Learn how to securely generate random numbers for your smart contract with Chainlink VRF (an RNG). This guide uses Solidity code examples.

Chainlink Documentation

Bad Randomness

Soluiton: Chainlink VRF



Signature Replay

What is a signature?

ECDSA: Elliptic Curve Cryptography

Message	Private Key	Public Key	Signature

user0 can sign the **message** using the **private key** and get the **signature**.

user1 can recover the **public key** given the **message** and the **signature**

A **public key** can be generated given a **private key**

Signature Replay

EVM Utils Demo

ERC 191 Signing

Sign typed data using ERC-191 | Personal message signing

Signature Replay

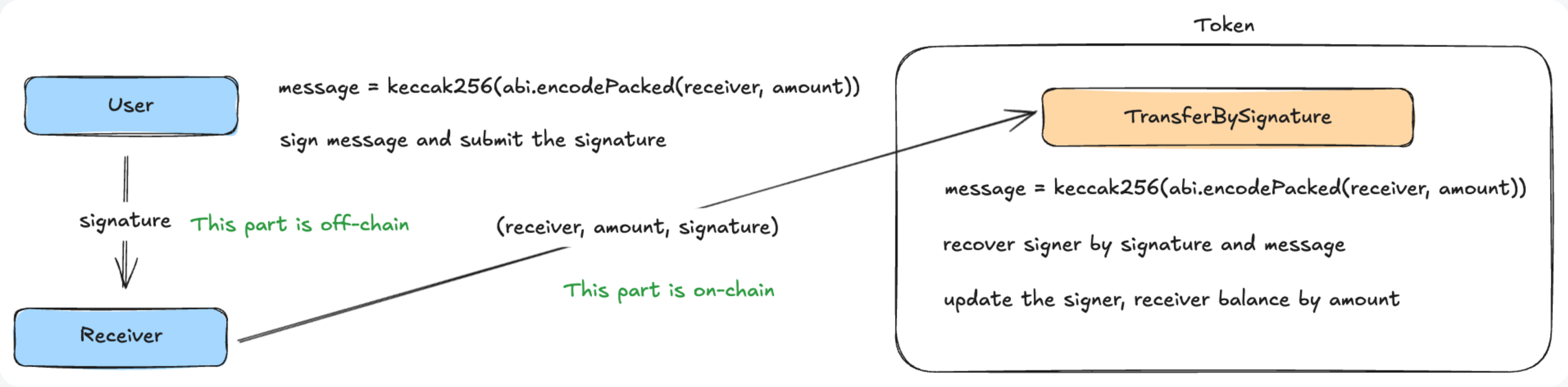
What can be a digest / message? Any 32 bytes data

- How to sign? Use metamask / Foundry cheatcode
- How to recover? Use ecrecover precompile address. - **NOTE**: return zero address(0) when the recover process fail.

```
(address alice, uint256 alicePk) = makeAddrAndKey("alice");
emit log_address(alice);
bytes32 hash = keccak256("Signed by Alice");
(uint8 v, bytes32 r, bytes32 s) = vm.sign(alicePk, hash);
address signer = ecrecover(hash, v, r, s);
assertEq(alice, signer); // [PASS]
```

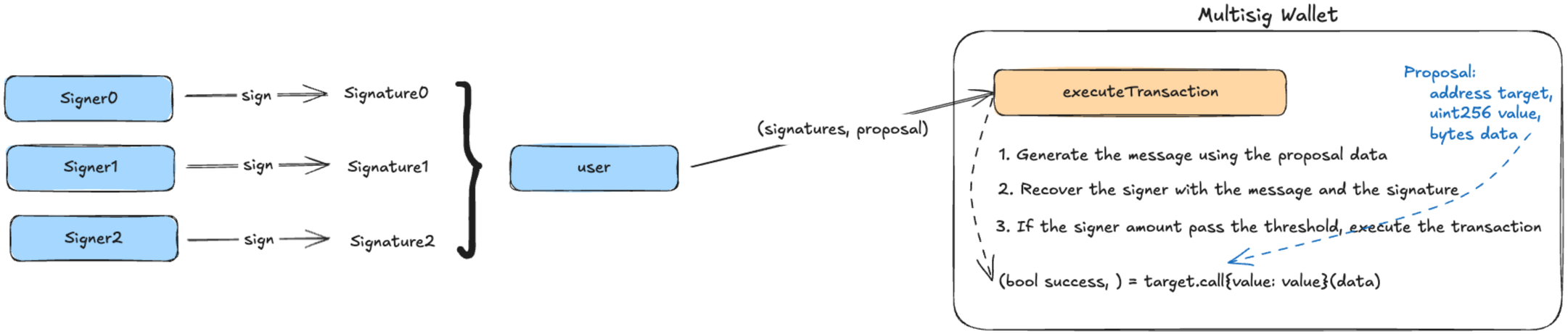
Signature Replay

What can this do?



Signature Replay

Multisig Wallet



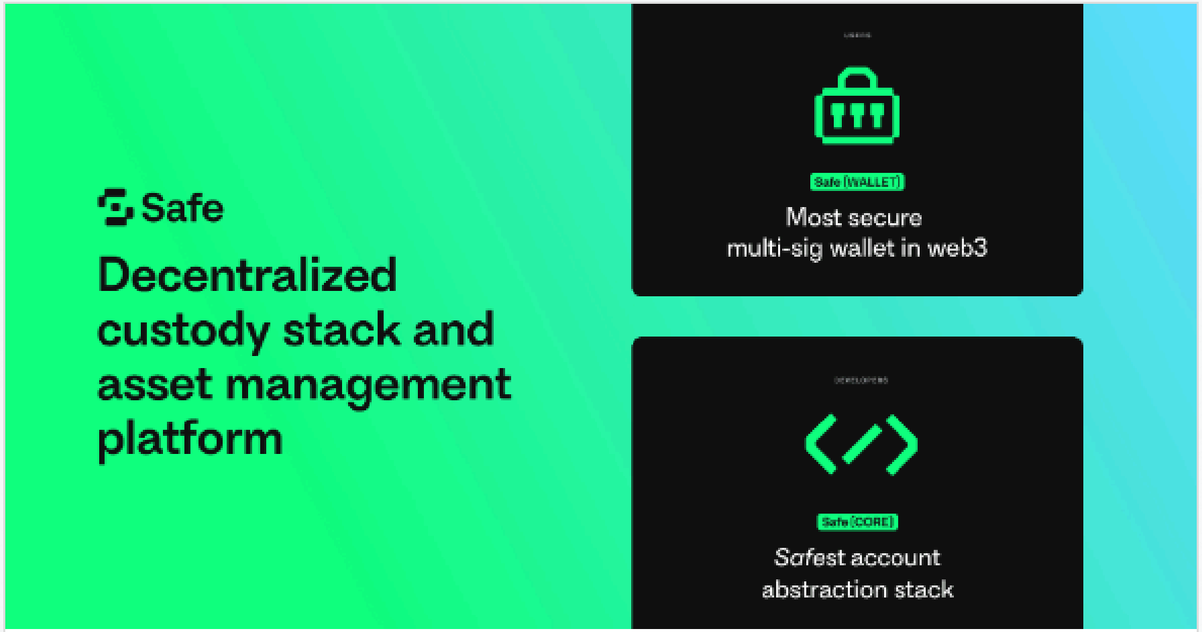
Signature Replay

Multisig Wallet

What might be the potential problem in this implementation?

Signature Replay

Real World Usage - Safe Wallet



Safe{Wallet}— Secure Smart Account Management on Ethereum

Explore Safe, the leading Ethereum multisig wallet for secure and efficient Smart Account management. Experience advanced security and user-friendly features.

Signature Replay

Some signature can be reused for multiple times.

- Scenario 1: A proposal will deposit 5 ether to lending protocol, and if some signers want to deposit again, can they re-submit the same signature?

- Scenario 2: A proposal is created on another EVM-compatible chain, it has been executed. The same wallet is created on the Ethereum mainnet, can the signature be re-used again?

Signature Replay

Some signature can be reused for multiple times.

- Adding a *nonce* value to your message, and verify whether this signature is used before.

- Adding a *chain.id* value to your message, verify whether this signature is used on another chain before.

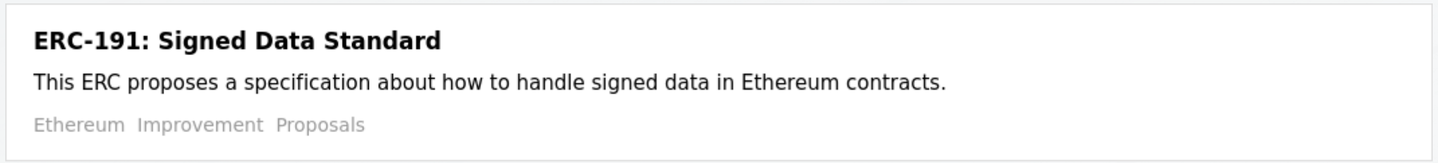
Signature Replay

EIP-191

0x19 <1 byte version> <version specific data> <data to sign>.

Signature Replay

EIP-712



0x19 <1 byte version> <version specific data> <data to sign>.

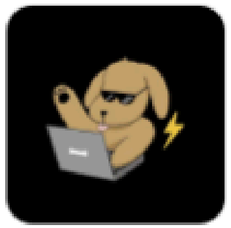
Signature Replay

ECDSA Oppenzeppelin Library

Smart Contract Analysis Tool

DEMO REPO

LouisTsai-Csie/**COSCUP-2024-Demo**



1 Contributor　0 Issues　0 Stars　0 Forks

LouisTsai-Csie/COSCUP-2024-Demo

Contribute to LouisTsai-Csie/COSCUP-2024-Demo development by creating an account on GitHub.

GitHub

Tools Category

We categorize the smart contract code analysis tools into three categories.

STATIC ANALYZER

Scan smart contract code without executing it, identifying potential vulnerabilities and inefficiencies

FUZZING TOOLS

Generate and send random inputs to test the robustness and security of smart contracts.

FORMAL VERIFICATION

Mathematically prove the correctness of the logic against its specifications, ensuring it behaves as intended under all conditions.

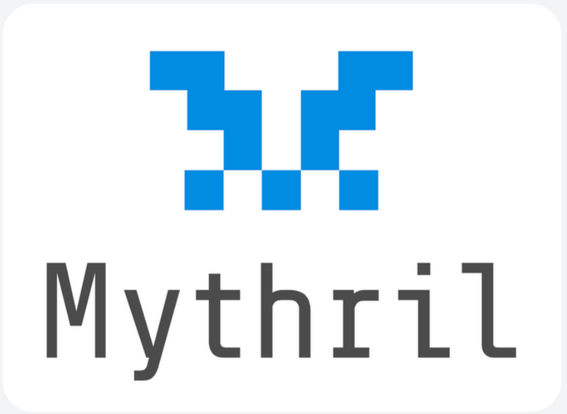
Demo: Static Analyzer

Scan smart contract code without executing it, identifying potential vulnerabilities and inefficiencies

Slither / Slitherin

Aderyn

Mythril



NOTE: After the Dencun upgrade, `selfdestruct` can only eliminate bytecode deployed during the contract creation transaction.

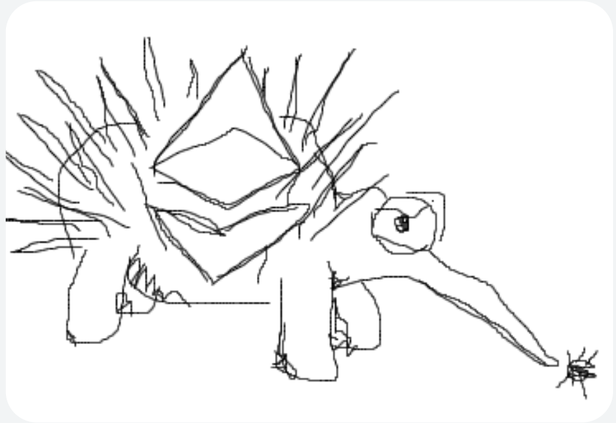
Demo: Fuzzing Tools

Generate and send random inputs to test the robustness and security of smart contracts.

Medusa

Foundry

Echidna



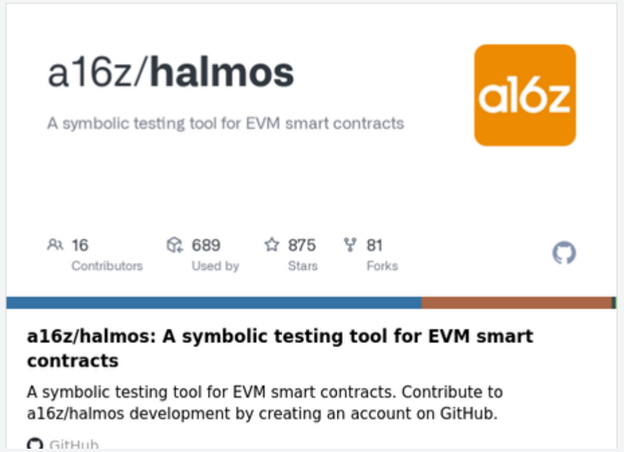
Demo: Formal Verification

Mathematically prove the correctness of the logic against its specifications, ensuring it behaves as intended under all conditions.

ItyFuzz

Certora

Halmos



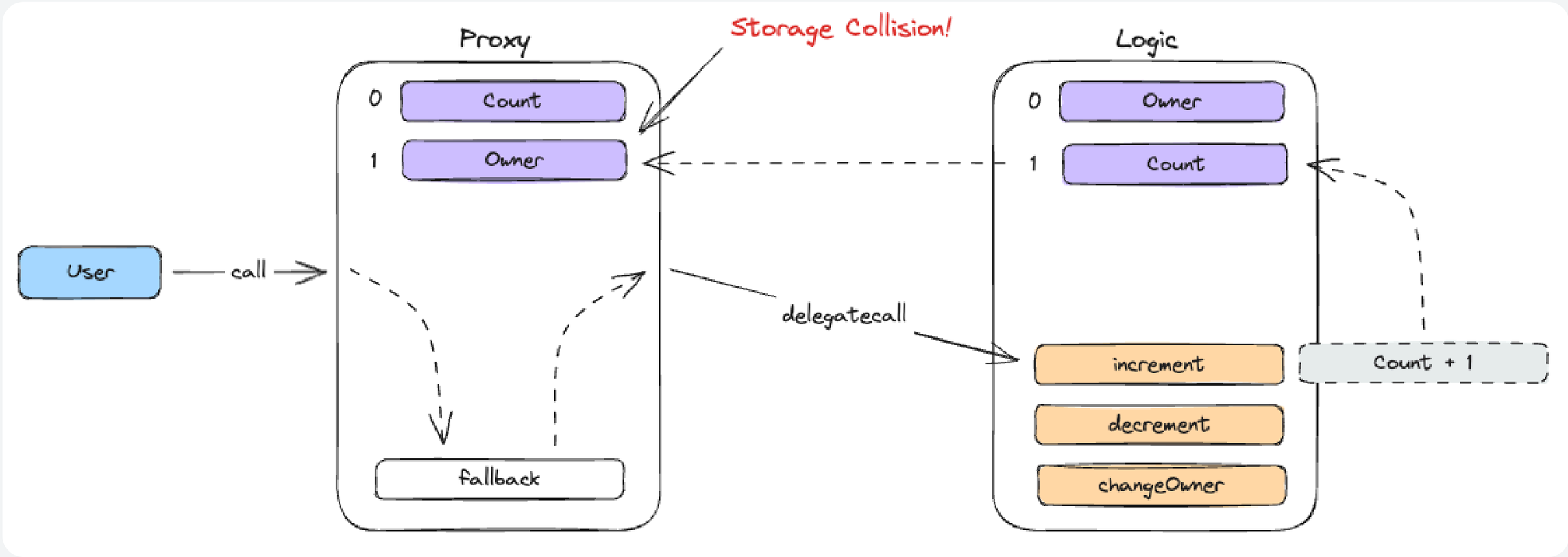
Awesome list

Awesome-Smart-Contract-Analysis-Tools

- Including 40+ Tools (focus on Solidity)
- Category: Static Analyzer, Fuzzing, formal verification tools, ...
- Keep adding new tools now!
- Submit your PR & Issues

Storage Collision

The inconsistency of the storage layout between proxy and implementation contract leads to storage collision vulnerability



Slither

Consistent Storage Layout: Ensure the storage layout of the proxy contract matches that of the implementation contract.

Slither-check-upgradeability

7	`order-vars-contracts`	Incorrect vars order with the v2	High
8	`order-vars-proxy`	Incorrect vars order with the proxy	High

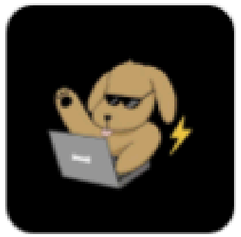
Demo Time

Audius Attack &
Slither-check-upgradeability

Attack Incident Reproduction

DEMO REPO

LouisTsai-Csie/**TEM-2024-Demo**



1 Contributor | 0 Issues | 1 Star | 0 Forks

LouisTsai-Csie/TEM-2024-Demo

Contribute to LouisTsai-Csie/TEM-2024-Demo development by creating an account on GitHub.

GitHub

Tracking Current and Past Attack Incidents

Incident Database

Includes all past attack incidents and analyses

Postmortem Analysis

Recent Attack Incident Analysis and Postmortem

Real Time Alert

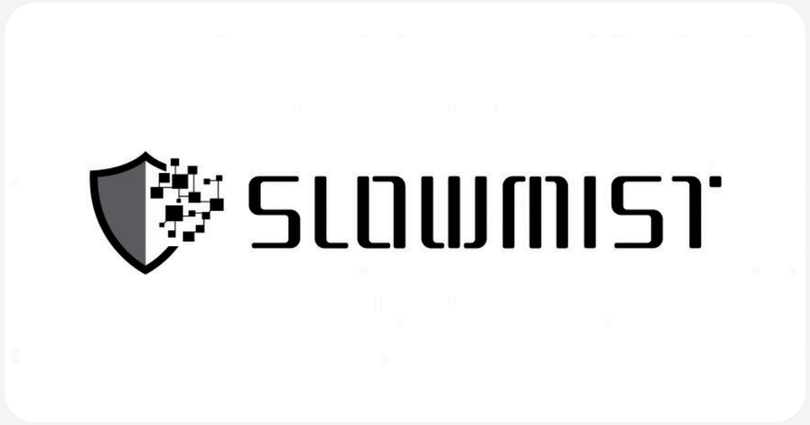
Reporting Ongoing Attack Incidents and Brief Analyses

Attack Incident Database

DeFiHackLabs

Rekt News

SlowMist



Phalcon

Postmortem Analysis

Rekt News : The dark web of DeFi journalism.

BlockThreat Intelligence : a weekly newsletter that provides the latest security updates, tools, events, vulnerabilities, and threats in the crypto landscape.

SlowMist Weekly Security Digest : Weekly attack incident overview in X (Twitter)

ChainLight Hack Digest : Weekly attack incident overview in X (Twitter).

BlockSec Monthly Security Review : Monthly attack incident overview.

Real Time Alert

Twitter is your good friend

0 PeckShield Alert

1 Phalcon Alert

2 Cyvers Alert

3 Certik Alert

4 Nic Franlin

5 Beosin Alert

6 MetaTrust Alert

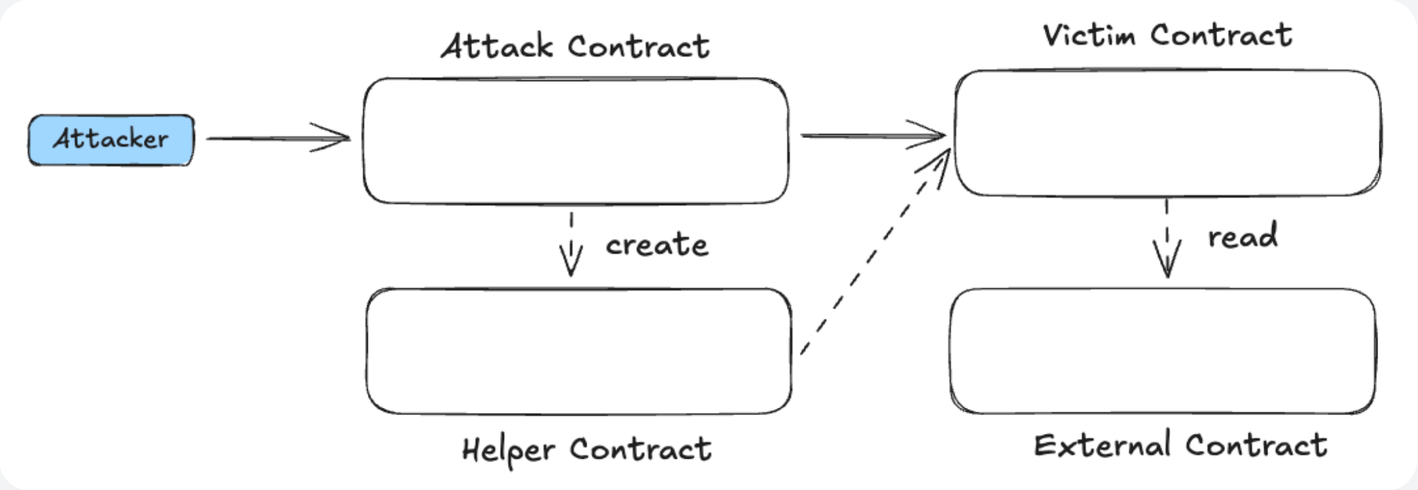
7 SlowMist Alert

#security-alert channel in DeFiHackLabs discord

Transaction Analysis

ROLE

Attacker, Attack Contract, Victim Contract, ...



If the transaction is complicated:

1. There might be some duplicate pattern
2. Collapse the staticcall section first

Attack Preparation

Request Flash Loan, Create Helper Contracts, ...

Exploit Procedure

Interact with the victim / external contract, ...

Post Attack Handling

Repay flash loan, transfer to wallet, ...

Transaction Analysis Tools

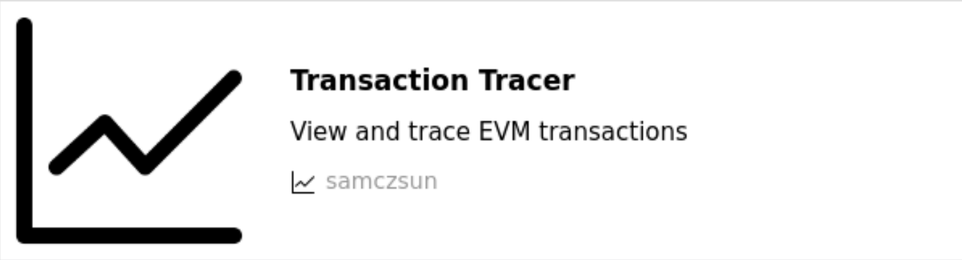
Tenderly

tenderly

AnyTx

Token Flow

Openchain



Phalcon

Demo!

Demo 0: FILX

Access Control

FLIX

Root Cause

The contract lacks a protected initialization function, allowing an attacker to reinitialize it, take over owner role, and subsequently withdraw all funds from the treasury.

Loss	~200K	PoC	Link
Chain	Ethereum	Transaction	Link

Attack Preparation

None

Exploit Procedure

Call the `init` function and take over the owner role

Post Attack Handling

swap FLIX token for USDT

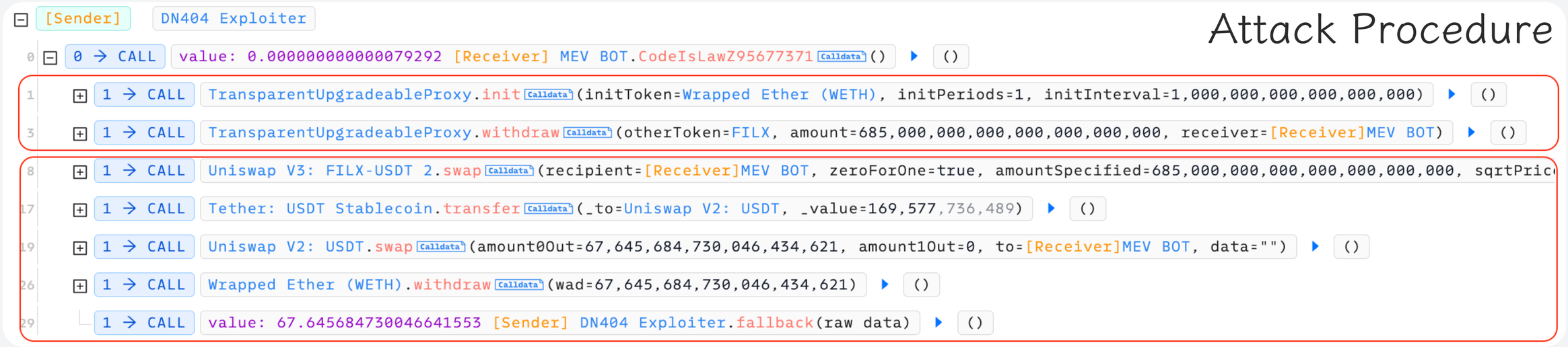
FLIX

```
51      /**
52       * @dev constructor
53       * @param initToken The t
54       */
55      function init(
56          IERC20 initToken,
57          uint256 initPeriods,
58          uint256 initInterval
59      ) public {
60          token = initToken;
61          periods = initPeriods;
62          interval = initInterval;
63          owner = _msgSender();
64      }
```

```
function withdraw(
    IERC20 otherToken,
    uint256 amount,
    address receiver
) public virtual onlyOwner {
    uint256 currentBalance = otherToken.balanceOf(address(this));
    require(receiver != address(0), "receiver must not empty");
    require(currentBalance >= amount, "current balance insufficient");
    otherToken.safeTransfer(receiver, amount);
}
```

The owner has been reset, allowing the `onlyOwner` modifier to be bypassed

FLIX



Attack Procedure

Post Attack Handling

Transaction

Demo 0: RICO

Arbitrary Call

RICO

Root Cause

An attacker can provide a target address and calldata to the Rico flash loan function, enabling the generation of calldata that directs Rico to transfer all funds to the attacker.

Loss	~36K	PoC	Link
Chain	Arbitrum	Transaction	Link

Attack Preparation

None

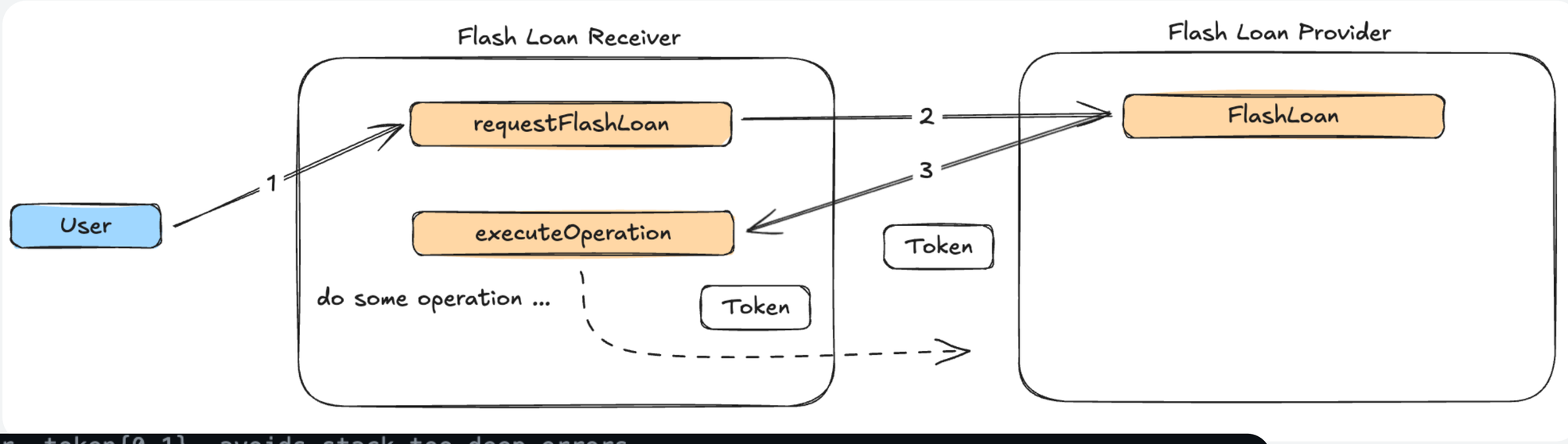
Exploit Procedure

Execute a malicious flash loan that forces Rico to transfer tokens out of the treasury.

Post Attack Handling

Swap token to USDC

Flash Loan Pattern




```
{ // scope for _token{0,1}, avoids stack too deep errors
address _token0 = token0;
address _token1 = token1;
require(to != _token0 && to != _token1, 'UniswapV2: INVALID_TO');
if (amount0Out > 0) _safeTransfer(_token0, to, amount0Out); // optimistically transfer tokens
if (amount1Out > 0) _safeTransfer(_token1, to, amount1Out); // optimistically transfer tokens
if (data.length > 0) IUniswapV2Callee(to).uniswapV2Call(msg.sender, amount0Out, amount1Out, data);
balance0 = IERC20(_token0).balanceOf(address(this));
balance1 = IERC20(_token1).balanceOf(address(this));
}
```

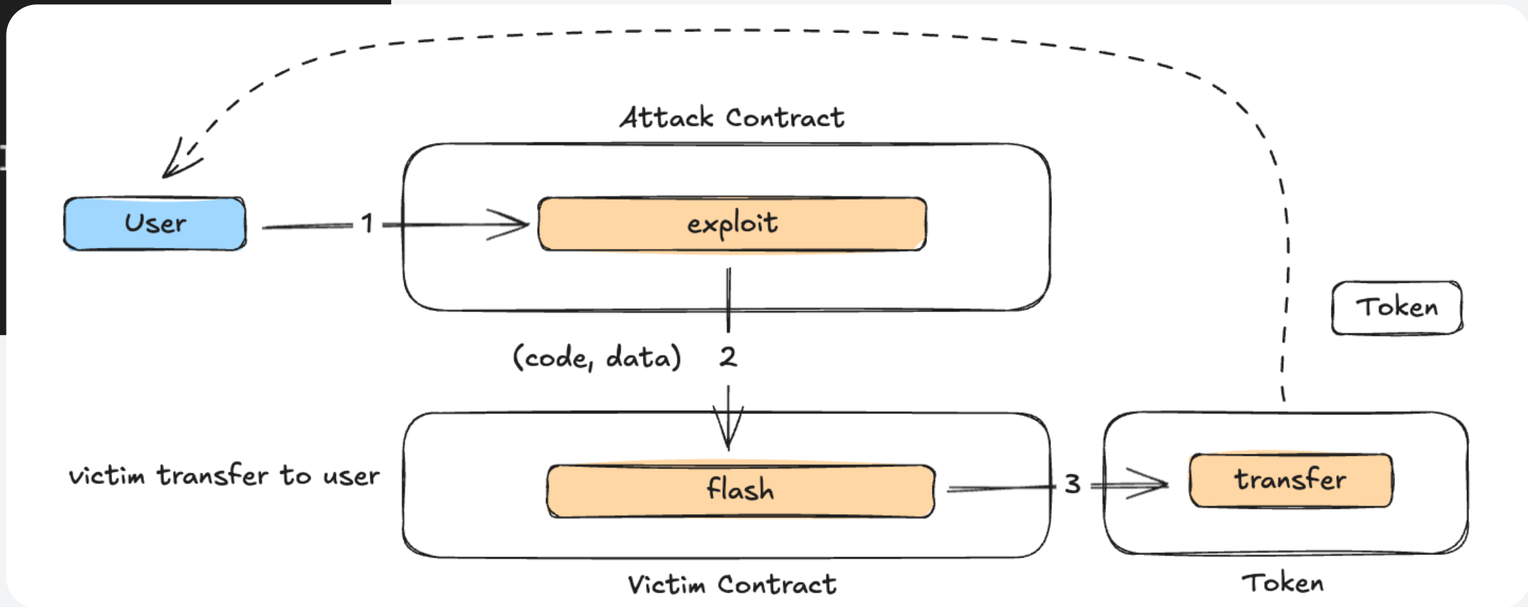
EIP-3156

Arbitrary Call

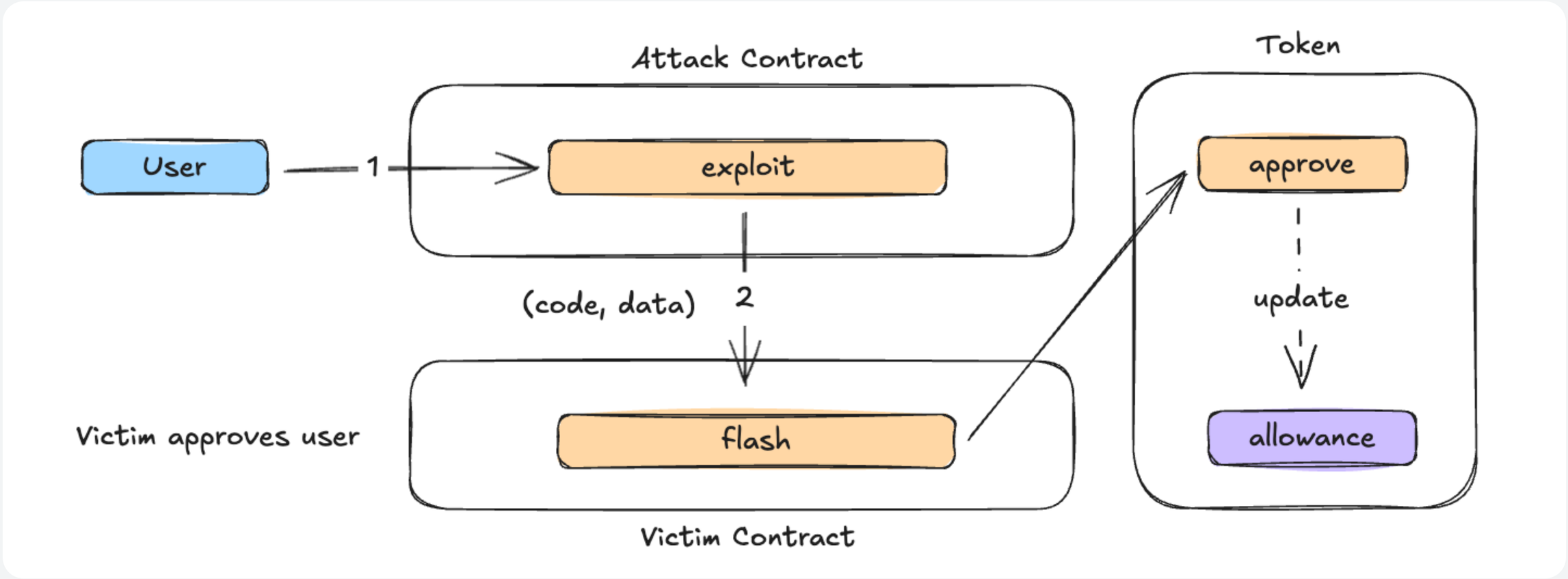
```
// flash borrow
// locked with itself to avoid flashing more than MINT
function flash(address code, bytes calldata data)
  external payable returns (bytes memory result) {
    // lock->mint->call->burn->unlock
    VatStorage storage vs = getVatStorage();
    if (vs.flock == LOCKED) revert ErrLock();
    vs.flock = LOCKED;

    getBankStorage().rico.mint(code, _MINT);
    bool ok;
    (ok, result) = code.call(data);
    if (!ok) bubble(result);
    getBankStorage().rico.burn(code, _MI

    vs.flock = UNLOCKED;
}
```

Not follow EIP-3156

The attacker provides a target address and call data to the victim contract to execute malicious actions.

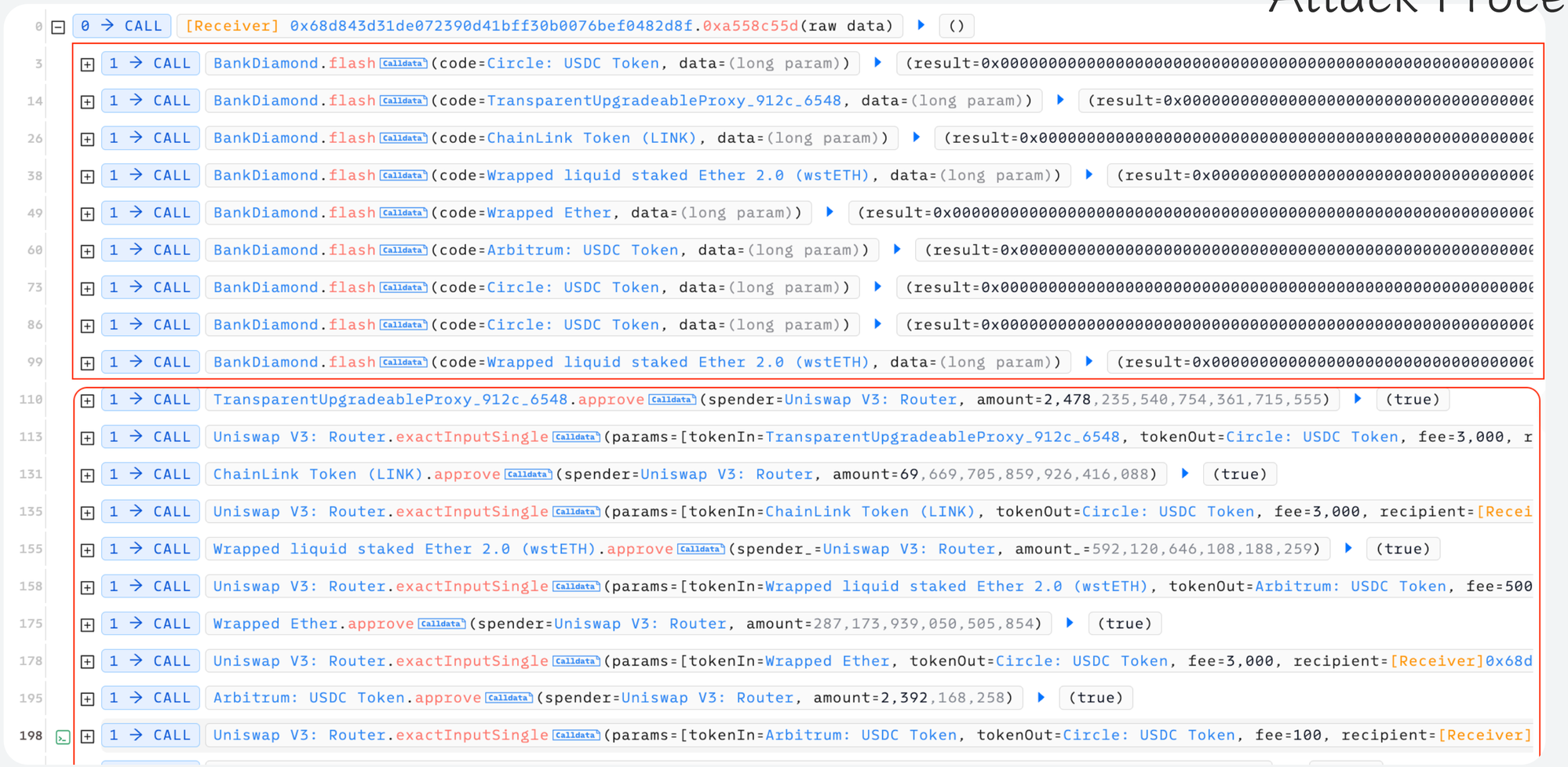


Arbitrary Call



Rico

Attack Procedure



Transaction

Post Attack Handling

Demo: GameToken

Reentrancy

GameToken

Root Cause

The newBidEtherMin() function contains a logic error that lets a bidder submit just over 5% of the previous bid, resulting in the full amount being returned to any last bid address.

_sendEther function use "call" to send ETH exposes the contract to re-entrancy attacks.

Loss	~20 ETH	PoC	Link
Chain	Ethereum	Transaction	Link

Attack Preparation

None

Exploit Procedure

Execute a malicious flash loan that forces Rico to transfer tokens out of the treasury.

Post Attack Handling

Swap token to USDC

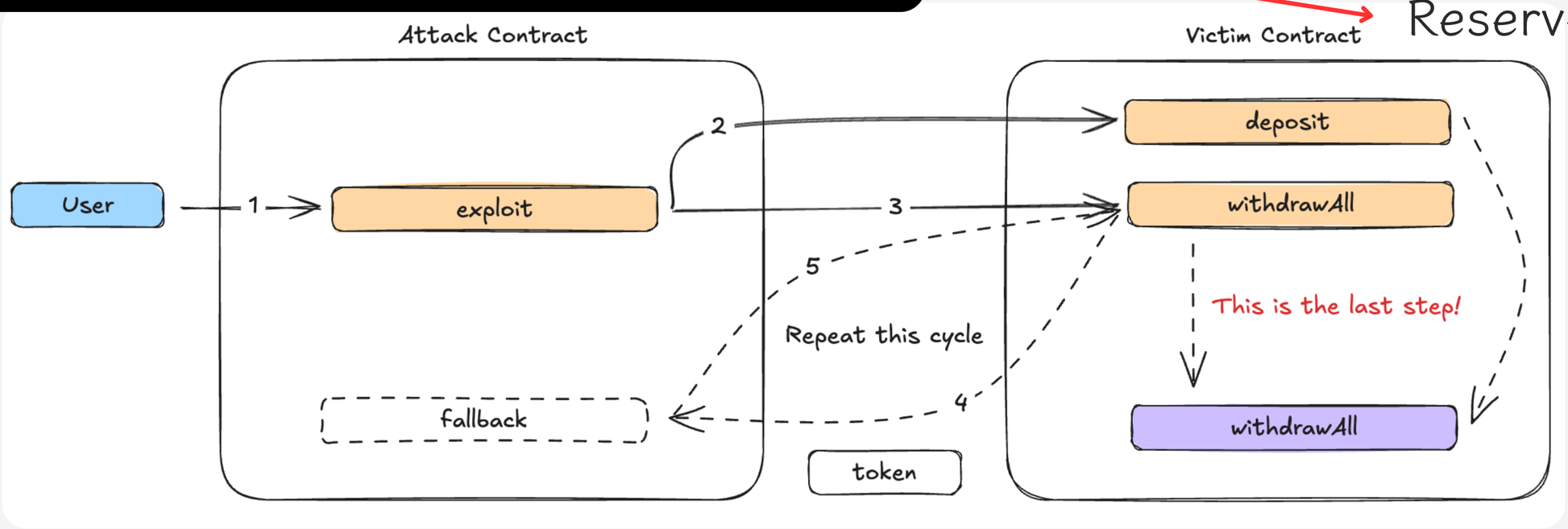
Reentrancy Issue

```
function withdrawAll() public {
    uint256 amount = _balances[msg.sender];
    (bool success,) = msg.sender.call{value: amount}("");
    require(success, "Ether Transfer Failed");

    _balances[msg.sender] = 0;

    emit unsafeBank__withdrawToken(msg.sender, amount);
}
```

Check-Effect-Interaction

1. Check: input validation
2. Effect: state variable update
3. Interaction: external contract call

Reserve Order



GameToken Attack

```
function newBidEtherMin() public view returns (uint256) {
    return (bidEther * auctionBidStepShare) / auctionBidStepPrecesion;
}
```

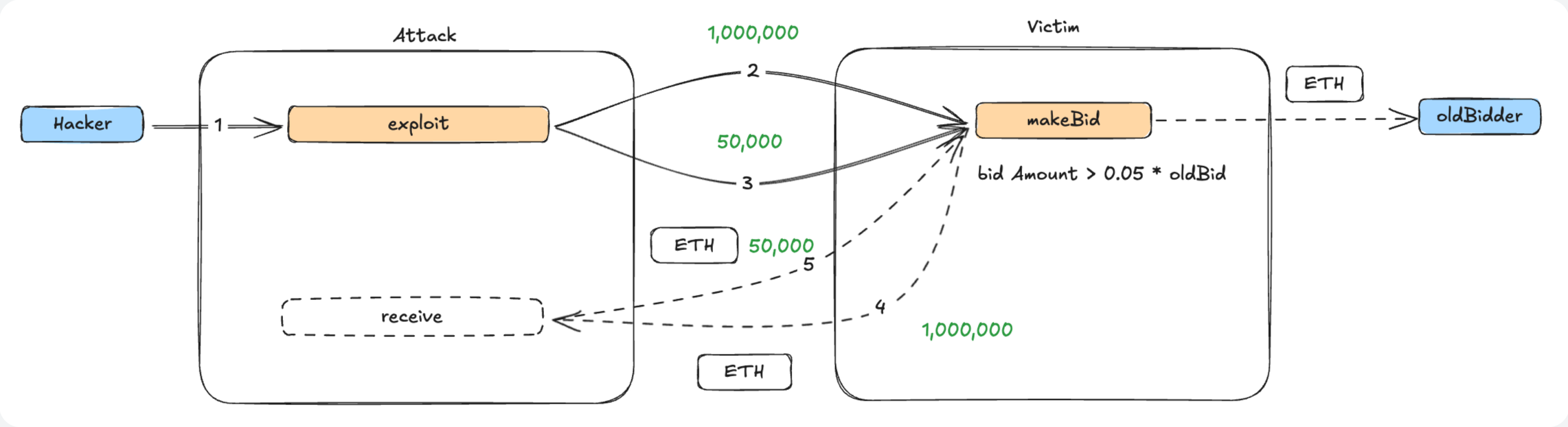
5

100

```
function makeBid() external payable {
    require(msg.value > newBidEtherMin(), "bid is too low");
    if (bidAddress != address(0)) {
        _sendEther(bidAddress, bidEther);
    }
    bidAddress = msg.sender;
    bidEther = msg.value;
    if (auctionEndTime == 0)
        auctionEndTime = block.timestamp + auctionStartTimer;
    else auctionEndTime += auctionBidAddsTimer;
}
```

```
function _sendEther(address to, uint256 count) internal {
    (bool sentFee, ) = payable(to).call{value: count}("");
    require(sentFee, "sent fee error: ether is not sent");
}
```

GameToken Attack



GameToken

Protection Method

1. Add reentrancy guard to the function to prevent recursive call
2. Follow the check-effect-interaction pattern

GameToken Attack

Transaction

Demo: GalaxyFoxToken

Access Control

GalaxyFoxToken

Root Cause

The setMerkleRoot function currently has no access controls, allowing anyone to reset the merkle root and subsequently claim rewards from the treasury.

The contract is unverified and requires decompilation

Loss	~50K	PoC	Link
Chain	Ethereum	Transaction	Link

Attack Preparation

No preparation for the attack

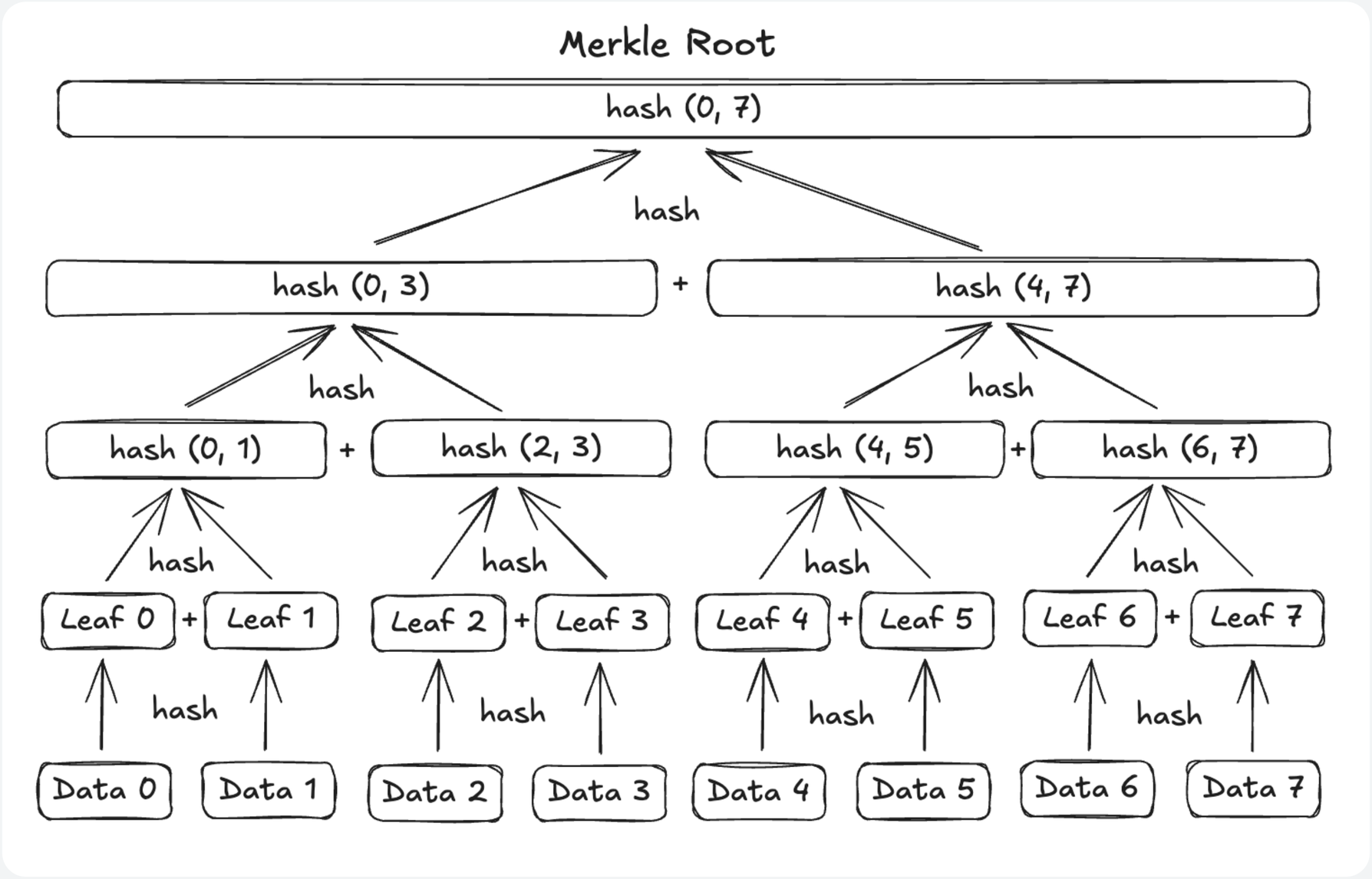
Exploit Procedure

Configure the merkle root

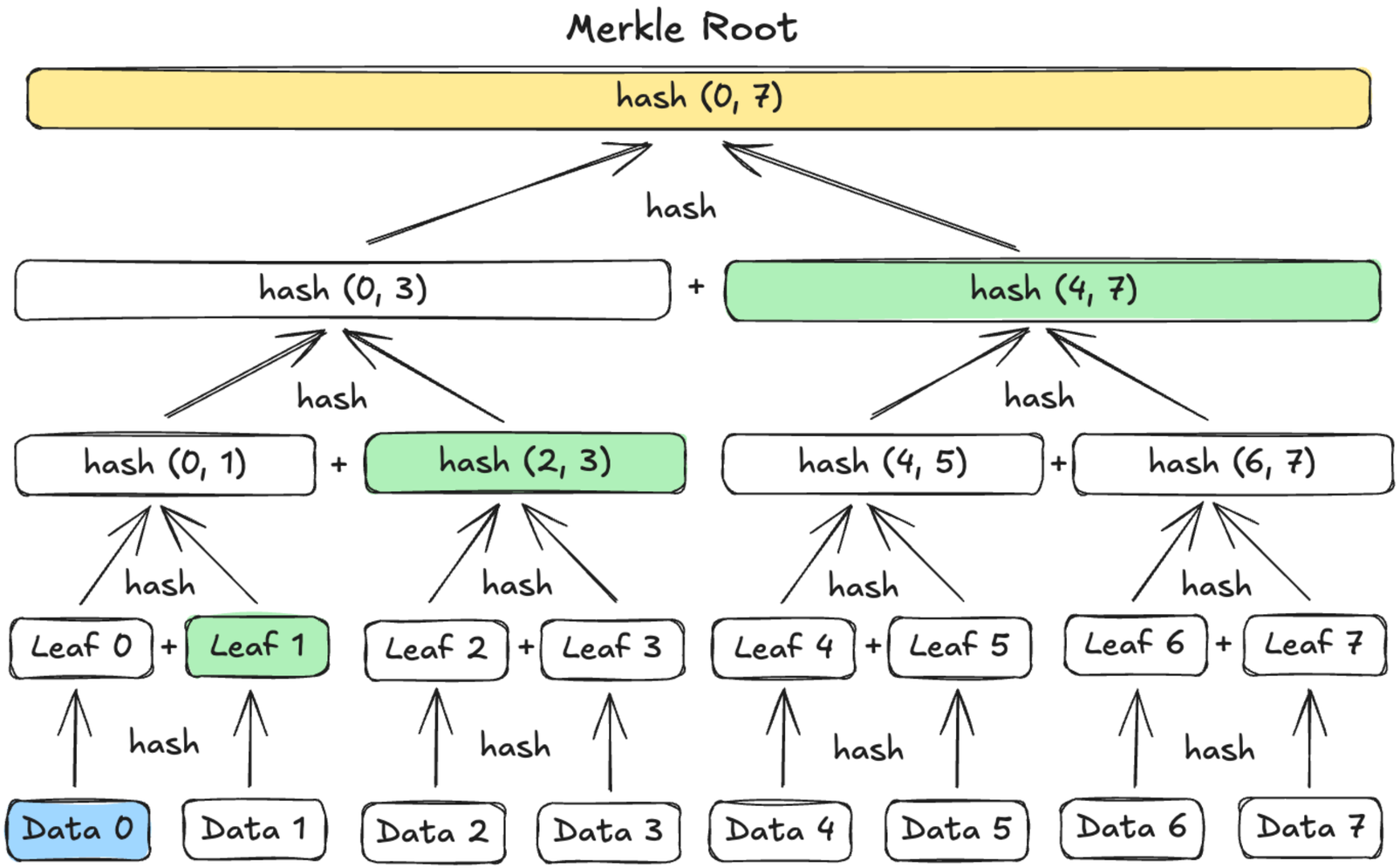
Post Attack Handling

Swap token to stable coin

Merkle Tree

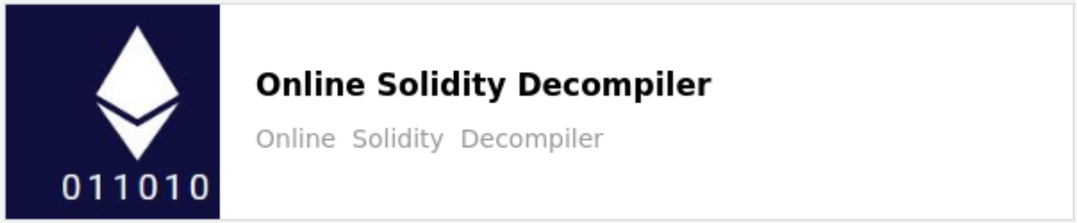

Merkle Root
hash (0, 7)
hash
hash (0, 3)
+
hash (4, 7)
hash
hash
hash (0, 1)
+
hash (2, 3)
hash (4, 5)
+
hash (6, 7)
hash
hash
hash
hash
Leaf 0
+
Leaf 1
Leaf 2
+
Leaf 3
Leaf 4
+
Leaf 5
Leaf 6
+
Leaf 7
hash
hash
hash
hash
Data 0
Data 1
Data 2
Data 3
Data 4
Data 5
Data 6
Data 7

Merkle Tree


Merkle Root
hash (0, 7)
hash
hash (0, 3)
+
hash (4, 7)
hash
hash
hash (0, 1)
+
hash (2, 3)
hash (4, 5)
+
hash (6, 7)
hash
hash
hash
hash
Leaf 0
+
Leaf 1
Leaf 2
+
Leaf 3
Leaf 4
+
Leaf 5
Leaf 6
+
Leaf 7
hash
hash
hash
hash
Data 0
Data 1
Data 2
Data 3
Data 4
Data 5
Data 6
Data 7

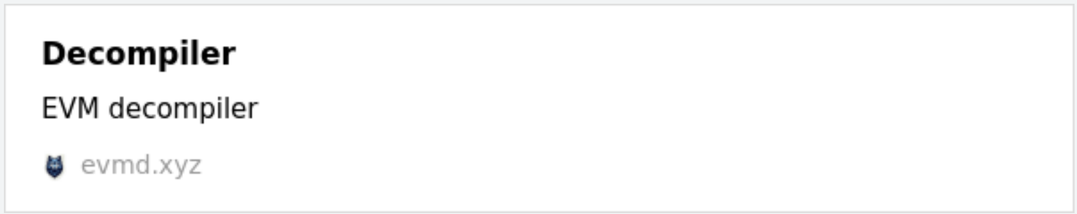
Decompilation Tools

The contract is unverified so you can not read the source code directly



Online Solidity Decompiler

Dedaub Decompiler



EVM Decompiler

GalaxyFoxToken

Access Control No protected modifier

call data length check

```
66    function setMerkleRoot(bytes32 _merkleRoot) public payable {
67        require(msg.data.length - 4 >= 32);
68        _merkleRoot = _merkleRoot;
69    }
```

```
21    function claim(address to, uint256 amount, bytes32[] proof) public payable {
22        require(msg.data.length - 4 >= 96);
23        require(proof <= uint64.max);
24        require(4 + proof + 31 < msg.data.length);
25        require(proof.length <= uint64.max);
26        require(4 + proof + (proof.length << 5) + 32 <= msg.data.length);
27        require(_claimStart > 0, Error('GfoxClaim: Not started'));
28        require(block.timestamp >= _claimStart, Error('GfoxClaim: Not started'));
29        require(amount > _claimedAmount[to], Error('GfoxClaim: Already claimed'));
```

Merkle Leaf Generation

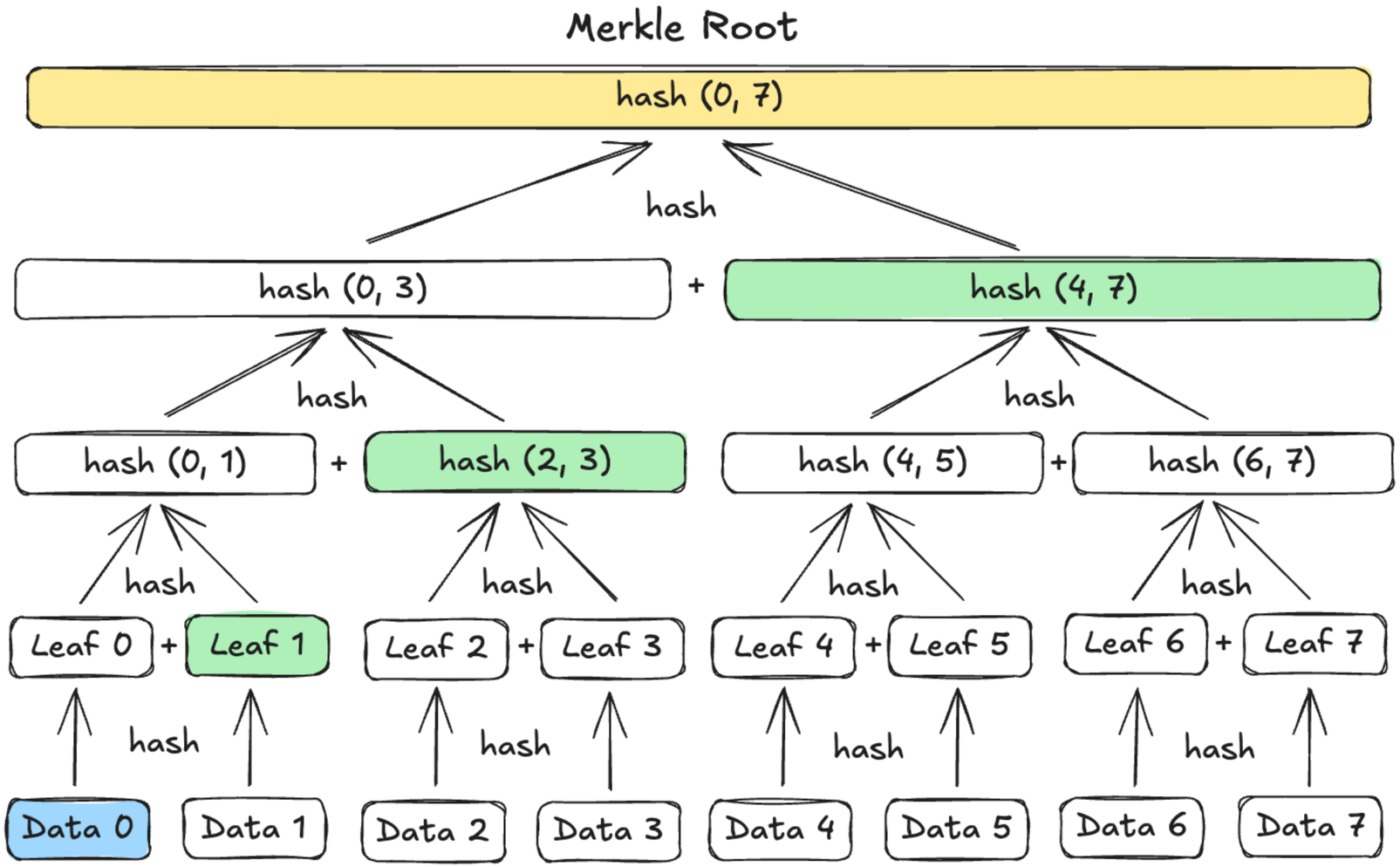
GalaxyFoxToken

Attack Preparation

Attack Procedure

Post Attack Handling

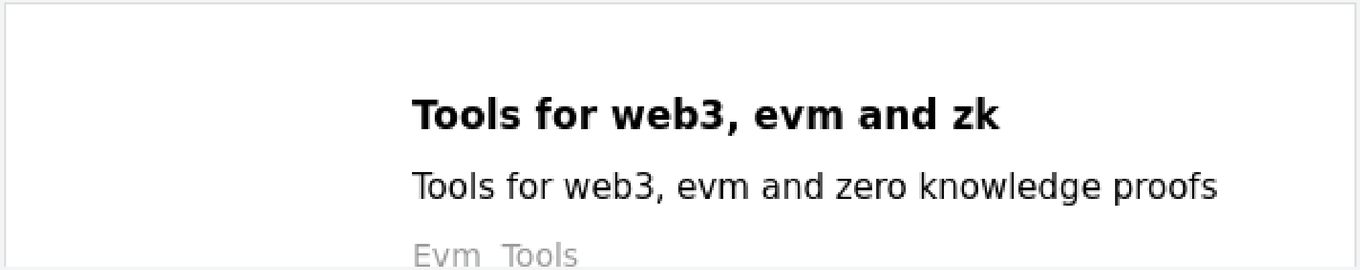
More on Merkle Tree



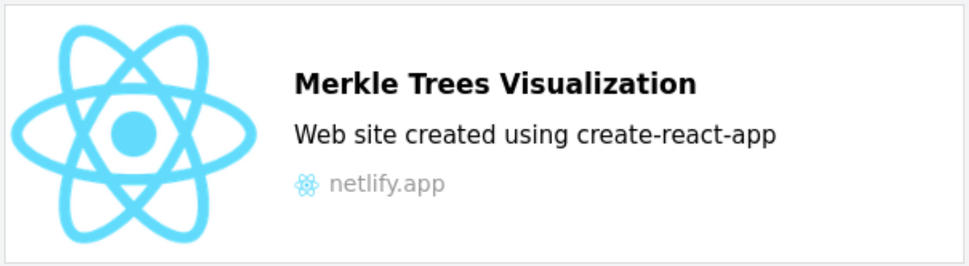
More on Merkle Tree

Merkle Tree Visualization

EVM Tools

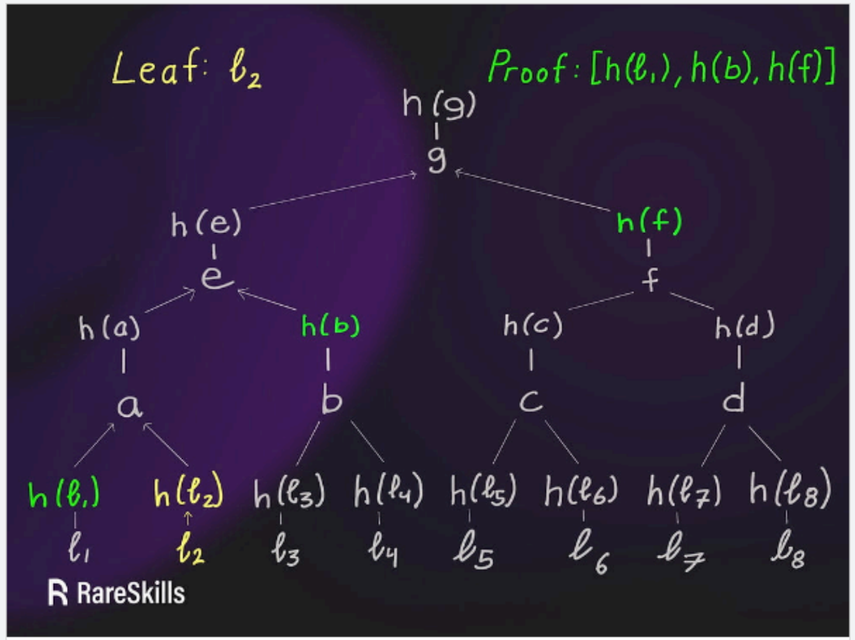


Merkle Tree Interaction



More on Merkle Tree

Please Please Please read this, or you cannot solve the challenge in your homework



The second preimage attack for Merkle Trees in Solidity

The second preimage attack in Merkle trees can happen when an intermediate node in a merkle tree is presented as a leaf.

RareSkills / Sep 17, 2024

More on Merkle Tree

Intermediate Node Vulnerability / Second Preimage Attack

- Make sure you cannot pass leaf value as input.
- If the function can pass leaf value, its length should not be 64 bytes
- Double hash to generate the leaf node given input parameter.

The End