Introduction to DeFi

09 / 27

CONTENT

Goals and Objectives

Objective 1 Understanding Upgradeable Proxy Concept & Implementation

Objective 2 Unveiling Security Issues in Upgradeable Proxy Patterns

Objective 3 Reproduce and Analyze Past Attack Incidents

Upgradeable Proxy Overview

Challenges in Smart Contract Development

- Due to the immutability of the blockchain system, you cannot change your smart contract after deployment, except for using `selfdestruct` operation*.

NOTE: `selfdestruct` cannot eliminate the deployed bytecode after the Dencun upgrade.

Challenges in Smart Contract Development

- Challenges -

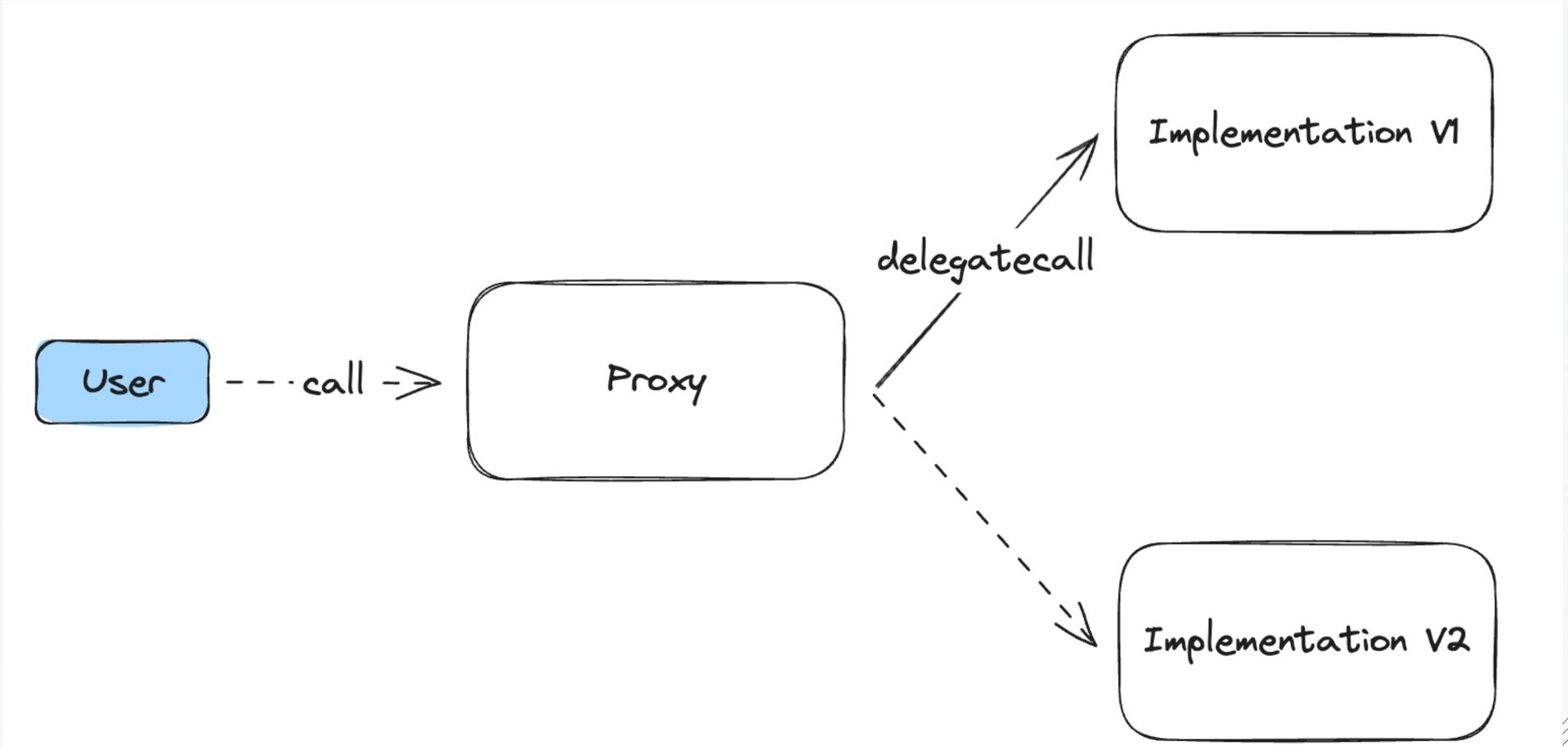
- What if there is a critical issue in your smart contract? How do you fix it without modifying the codebase?
- How do you add new features to your protocol? Redeploying a new contract would require users to change the address they interact with.

Challenges in Smart Contract Development

- Sloutions -

- The Upgradeable Proxy pattern consists of a **proxy** contract and an **implementation** contract.
- Users always interact with the proxy, and this pattern allows developers to **upgrade** the implementation part.

Upgradeable Proxy Overview



It looks great, but there are several security concerns that have led to significant losses.

EXPLOITS

Audius, EFVault, and Pike Finance have been affected by vulnerabilities related to upgradeable proxies, while Aave, Wormhole, and Teller have disclosed critical issues associated with proxies.

9M+ LOSS

10M+ BOUNTY

Unit: USD

Solidity, Foundry and EVM Recap

RECAP

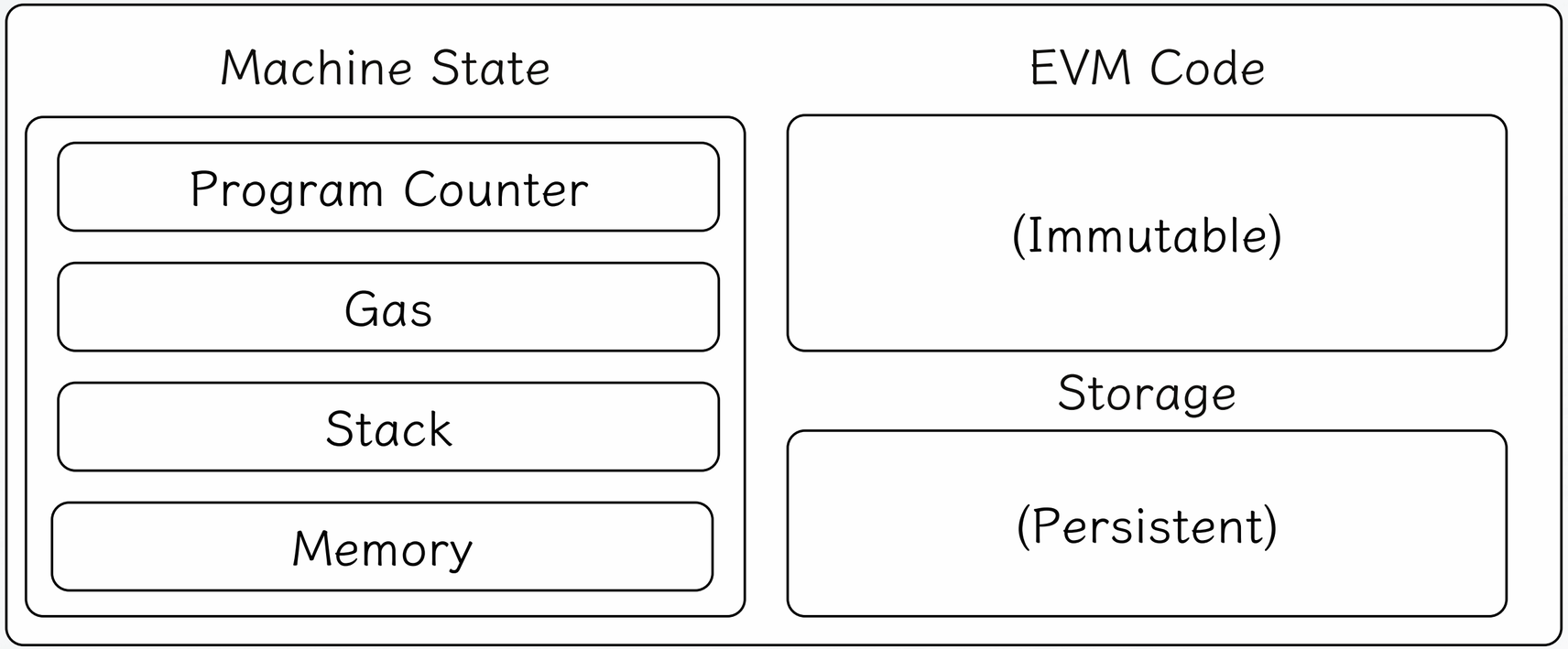
Before the workshop begins, we will introduce advanced topics in Solidity and EVM using the Foundry testing framework to help you better understand the concept of Upgradeable Proxy Security issues later on.

Storage Layout	How are state variables stored in the EVM's storage component?
Low-level Call	What are the differences between `call` and `delegatecall`, and how do they work?
Function Dispatching	What is the mechanism behind an external call, and how do you construct calldata?

Storage Layout

What is storage in the EVM, and how does it store state variables in this component?

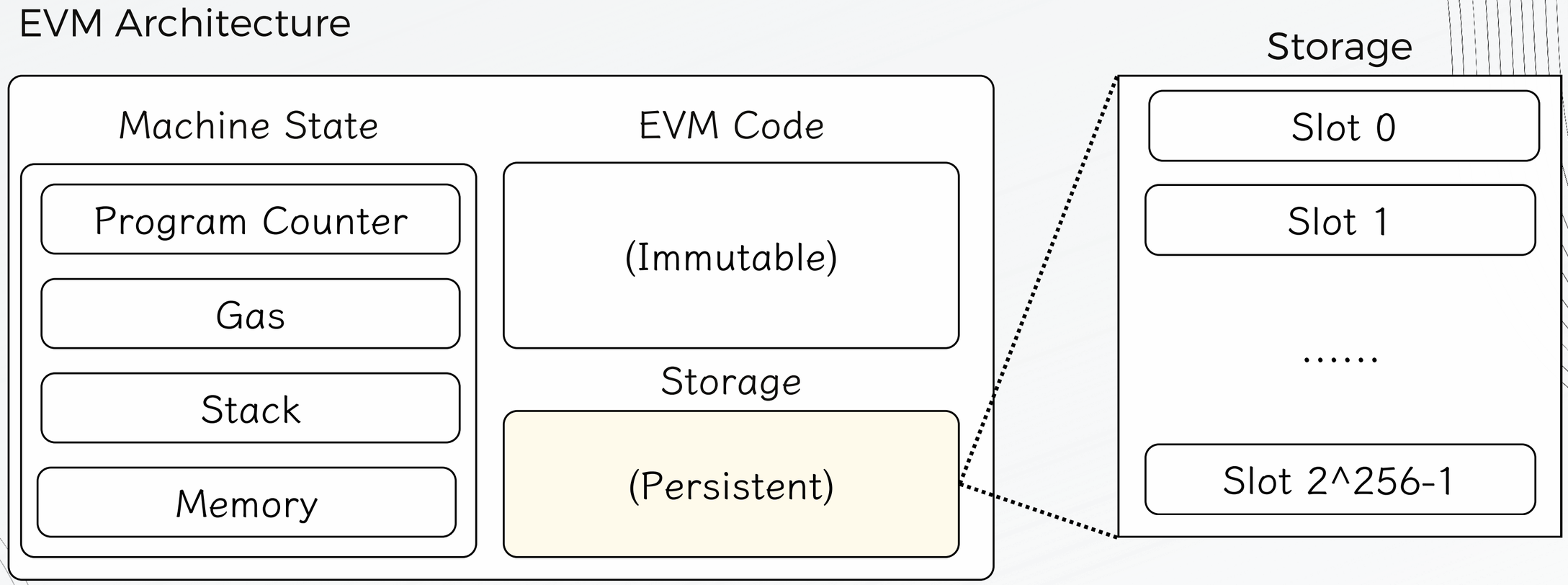
EVM Architecture



We have transient storage after the recent upgrade

Storage Layout

Storage slots starts from slot 0 to slot 2^256-1, with each slot storing 32 bytes.



LAB 1

STORAGE LAYOUT

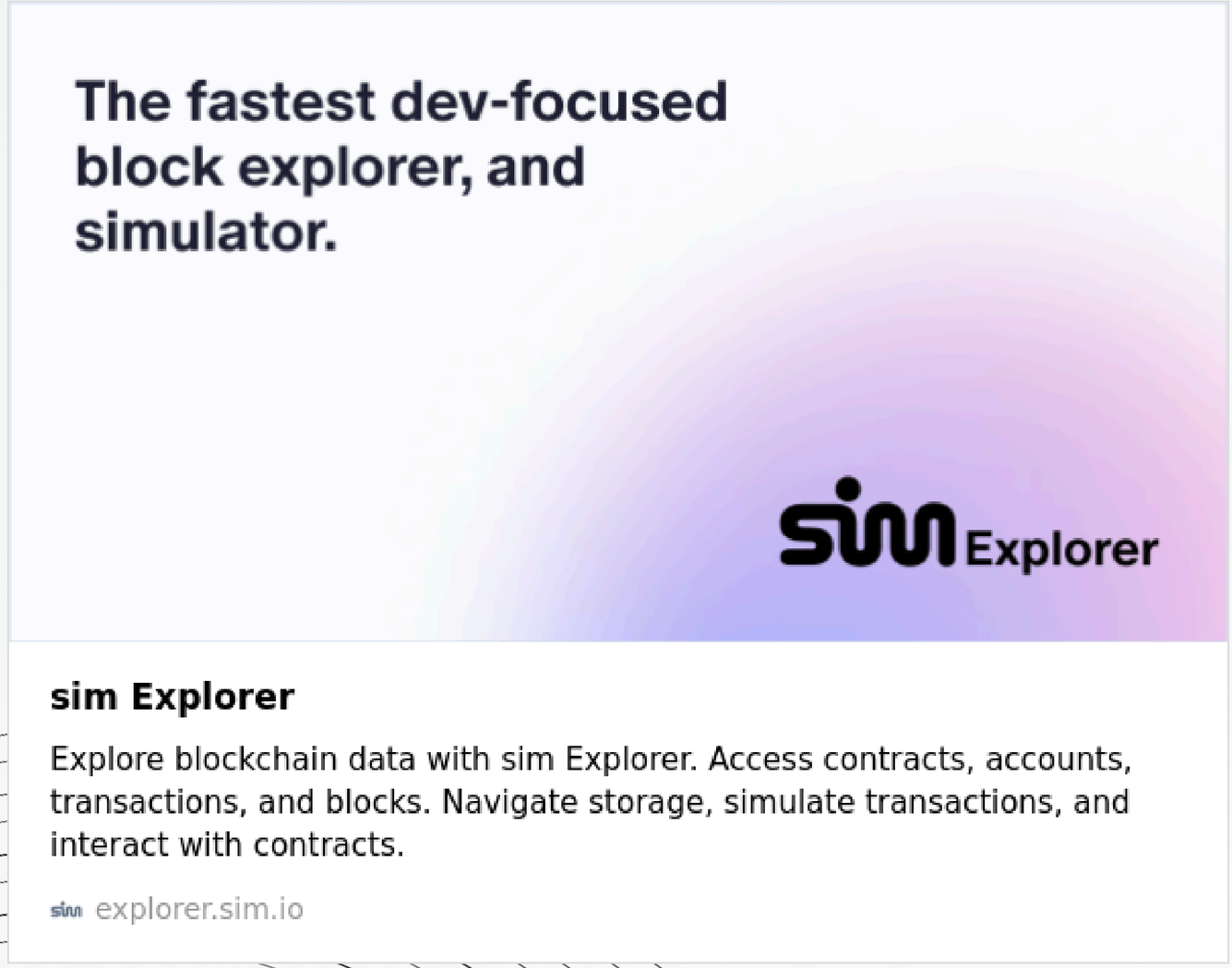
Storage Layout

Rules

- The first item in a storage slot is stored with lower-order alignment.
- Value types use only the necessary bytes for storage.
- If a value type exceeds the remaining space in a storage slot, it will be stored in the next slot.
- Structs and array data initiate a new slot and follow these packing rules.
- Following struct or array data, subsequent items always initiate a new storage slot.

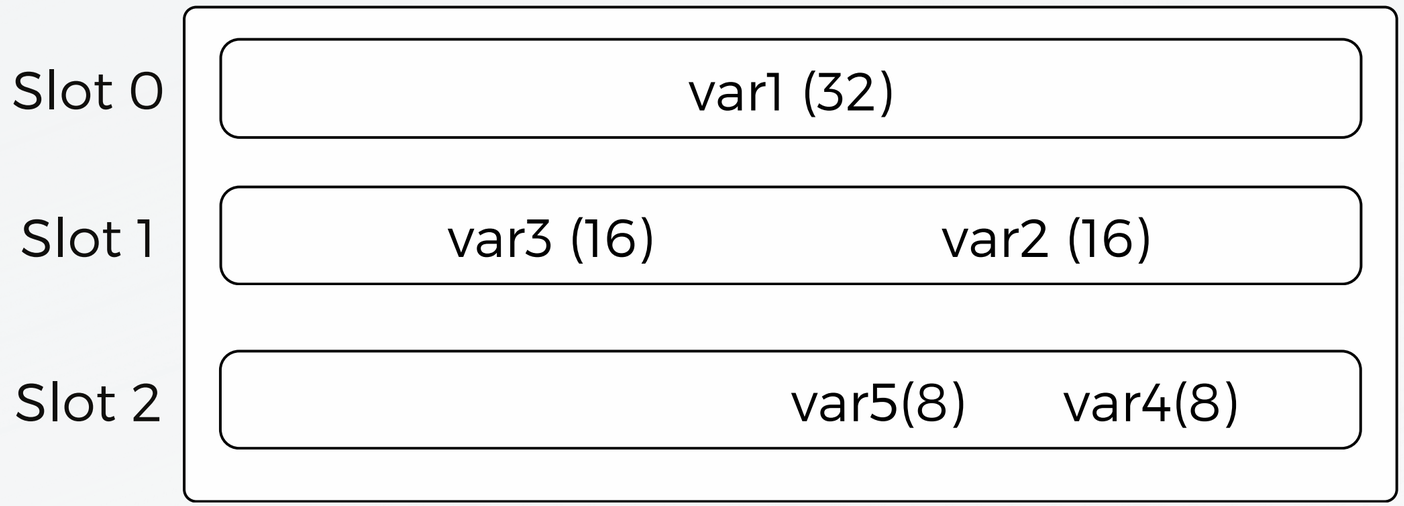
NOTE: Immutable and constant variables are not stored in storage but in the contract bytecode.

LAB 1


The fastest dev-focused block explorer, and simulator.
sim Explorer
sim Explorer
Explore blockchain data with sim Explorer. Access contracts, accounts, transactions, and blocks. Navigate storage, simulate transactions, and interact with contracts.
explorer.sim.io

Storage Layout Example #1

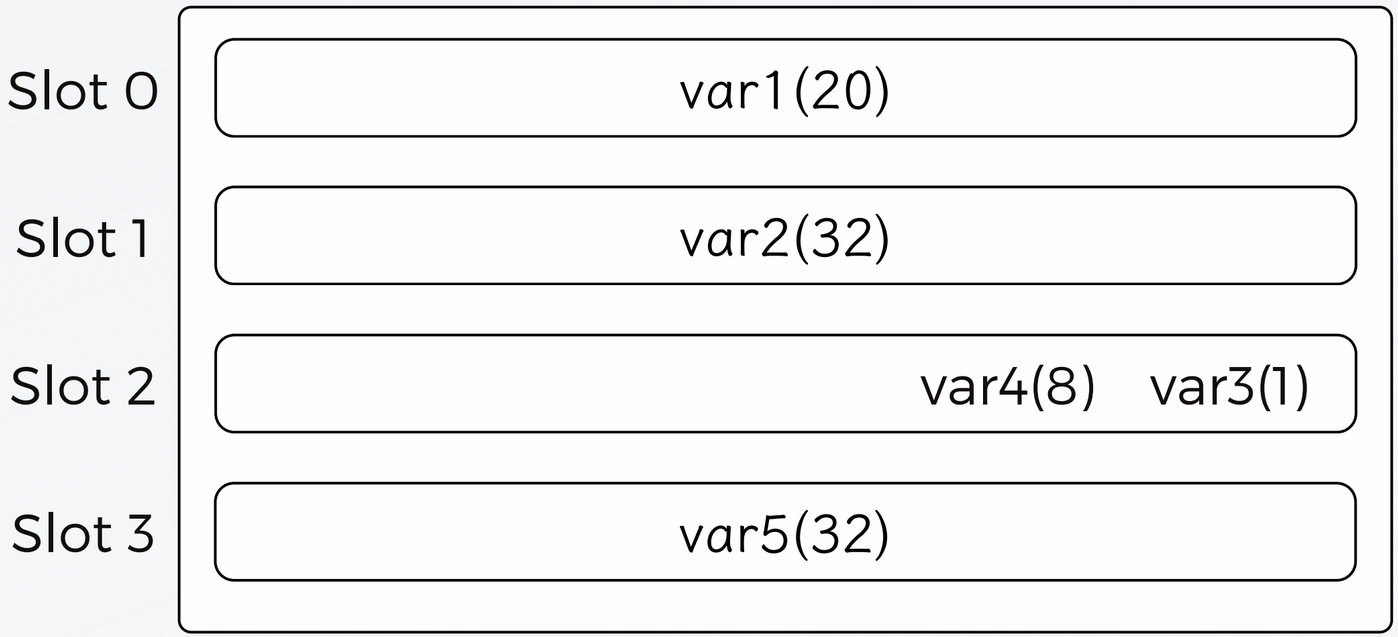
Please check the contract simpleStorageV1 in `demo/src/Storage.sol`



Command: forge inspect Storage.sol:simpleStorageV1 storage --pretty

Storage Layout Example #2

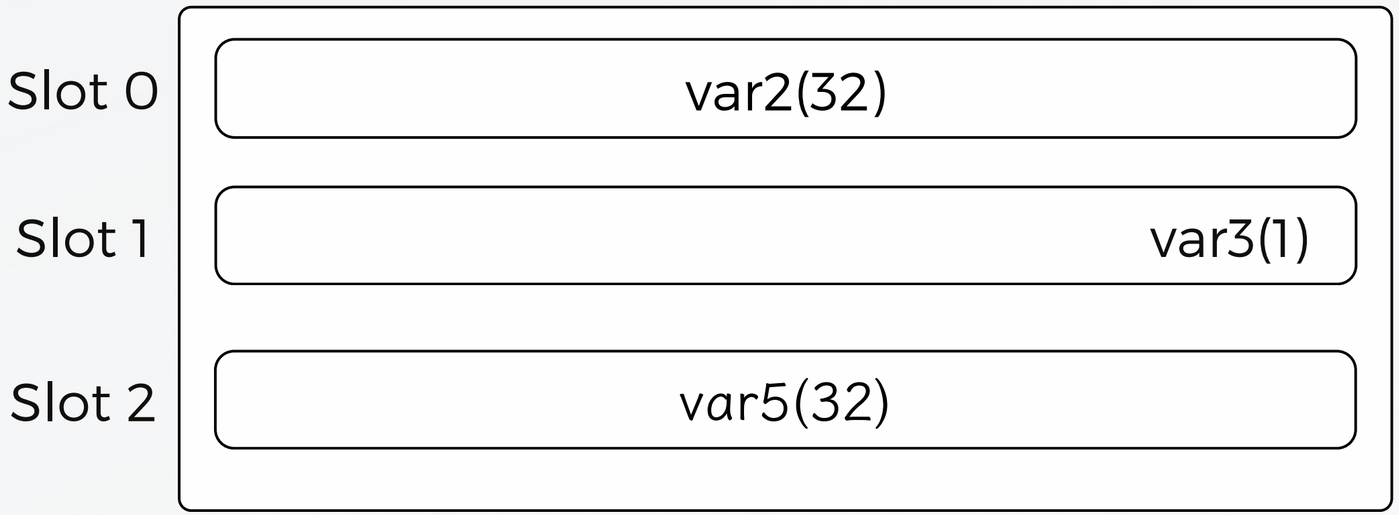
Please check the contract simpleStorageV2 in `demo/src/Storage.sol`



Command: forge inspect Storage.sol:simpleStorageV2 storage --pretty

Storage Layout Example #3

Please check the contract simpleStorageV3 in `demo/src/Storage.sol`



Command: forge inspect Storage.sol:simpleStorageV3 storage --pretty

LAB 2

LOW-LEVEL CALL

Low-level Call

Question: I have the `unsafeBank` contract address, how can I deposit my ethers through the deposit function?

Solution 1: Obtain the interface of unsafeBank and execute an external call.

-> IUnsafeBank(address).deposit{value: 1 ether}()

```
interface IUnsafeBank {
    function deposit() external payable;
    function withdraw(uint256 amount) external;
    function withdrawAll() external;
    function RUGPULL() external;
    function owner() external view returns (address);
    function balances(address addr) external view returns (uint256);
}
```

Low-level Call

Question: I have the `unsafeBank` contract address, how can I deposit my ethers through the deposit function?

Solution 2: Build the external calldata and transmit it to the target contract via a low-level call.

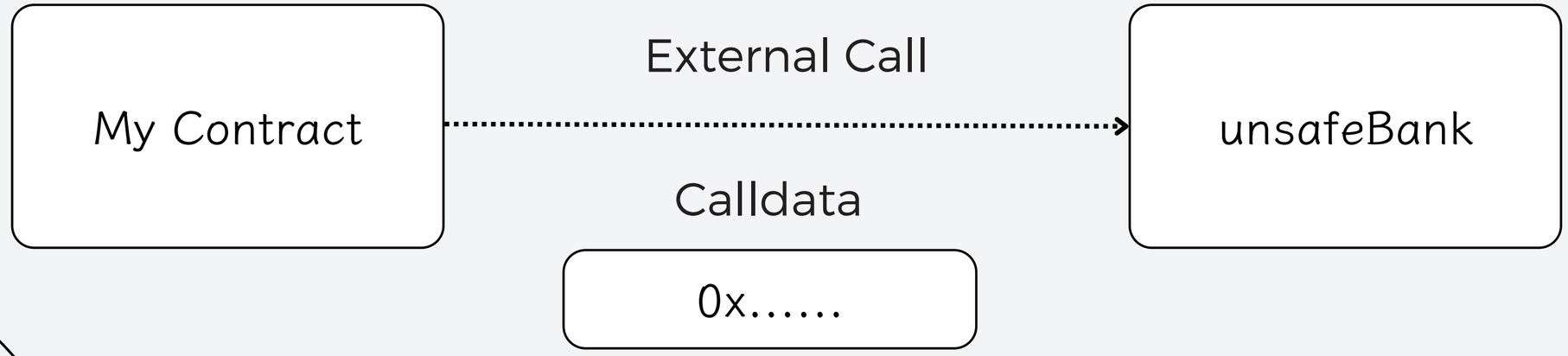
```
// External call using interface
IUnsafeBank(address(bank)).deposit{value: 1 ether}();

// External call using low-level call
bytes memory data = abi.encodeWithSignature("deposit()");
(bool success, ) = address(bank).call{value: 1 ether}(data);
require(success, "failed");
```

Low-level Call

Question: I have the `unsafeBank` contract address, how can I deposit my ethers through the deposit function?

Solution 2: Build the external calldata and transmit it to the target contract via a low-level call.



Function Dispatching

How to construct a calldata and make external call through low-level call?

Function	function balances(address addr) external returns(uint256)
Function Signature	balances(address)
Function Selector	bytes4(keccak256(“ balances(address)”))
Calldata	function selector + enc(function argument)*

*enc() follows the ABI-Encoding rules

Function Dispatching

How to construct a calldata and make external call through low-level call?

abi.encodeWithSignature — abi.encodeWithSignature("balances(address)", addr)

abi.encodeWithSelector — abi.encodeWithSignature(0x27e235e3, addr)

cast sig "balances(address)"

Function Dispatching

0x27...

=

function selector 0x27e235e3

+

Arguments

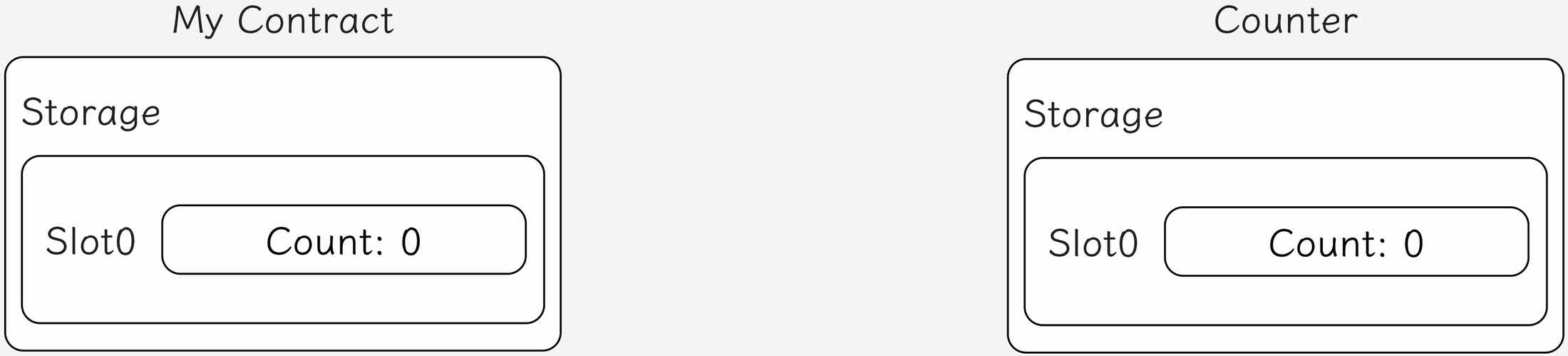
unsafeBank

0xd0e30db0	deposit
0x2e1a7d4d	withdraw
0x853828b6	withdrawAll
0x27e235e3	balances
0x8da5cb5b	owner
	fallback

LAB 3

DELEGATECALL & CALL

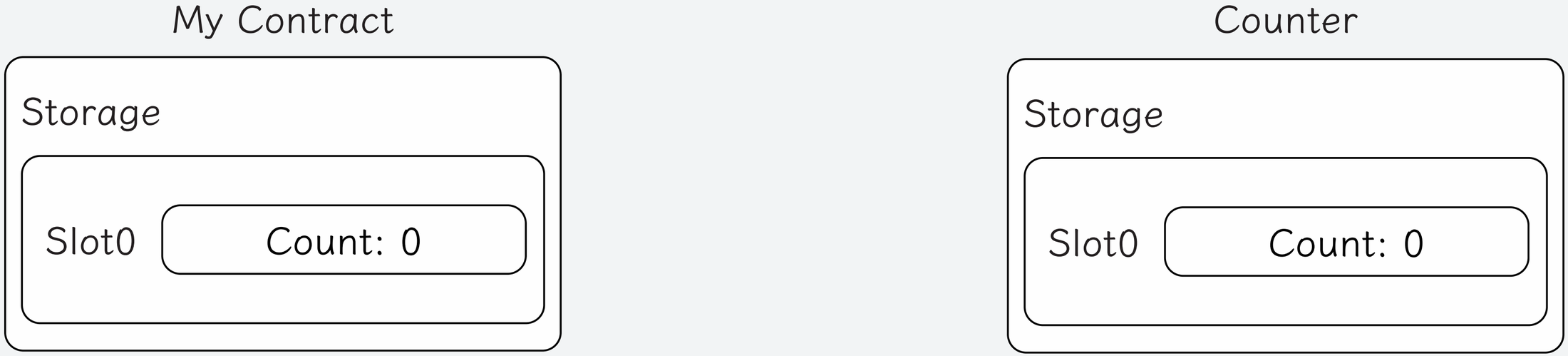
Distinct Types of Low-level Call



There is a function `add` in Counter
that will increase counter (slot0) value by 1

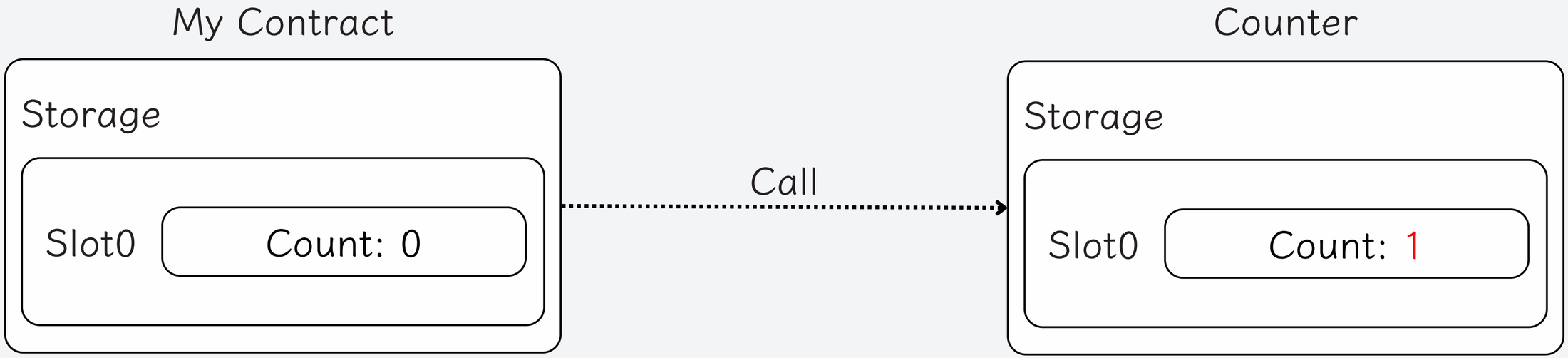
Demo

Call vs Delegatecall



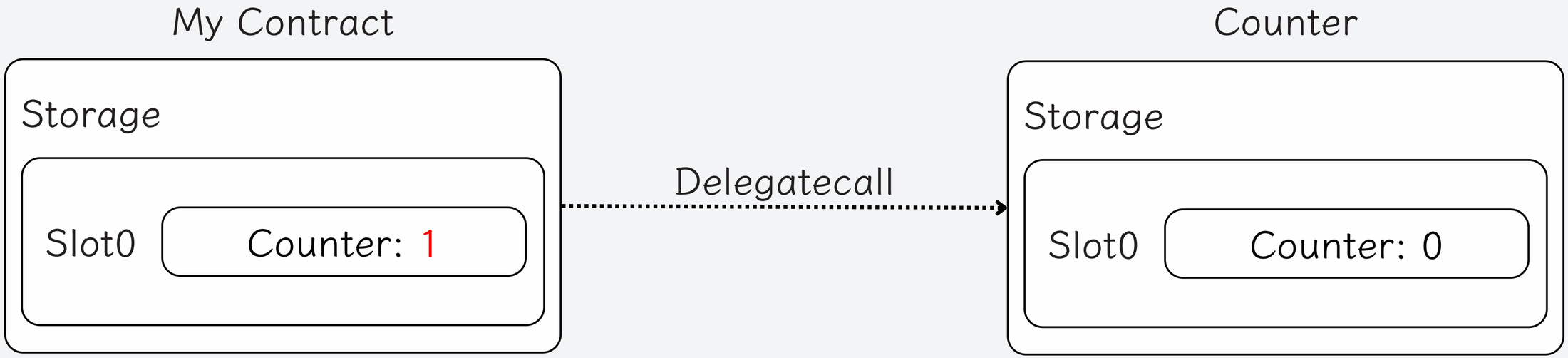
There is a function `add` in Counter
that will increase counter (slot0) value by 1

Call vs Delegatecall



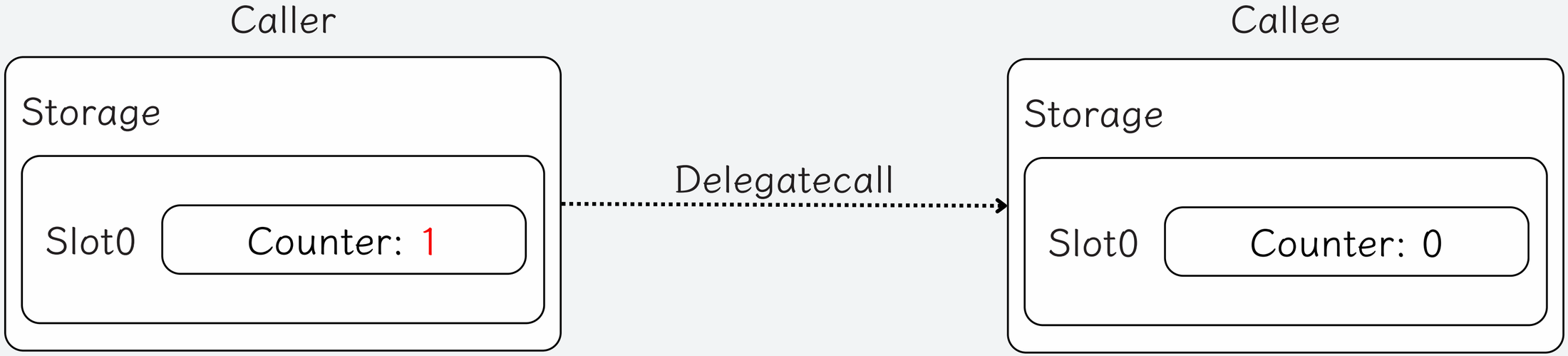
There is a function `add` in Counter
that will increase counter (slot0) value by 1

Call vs Delegatecall



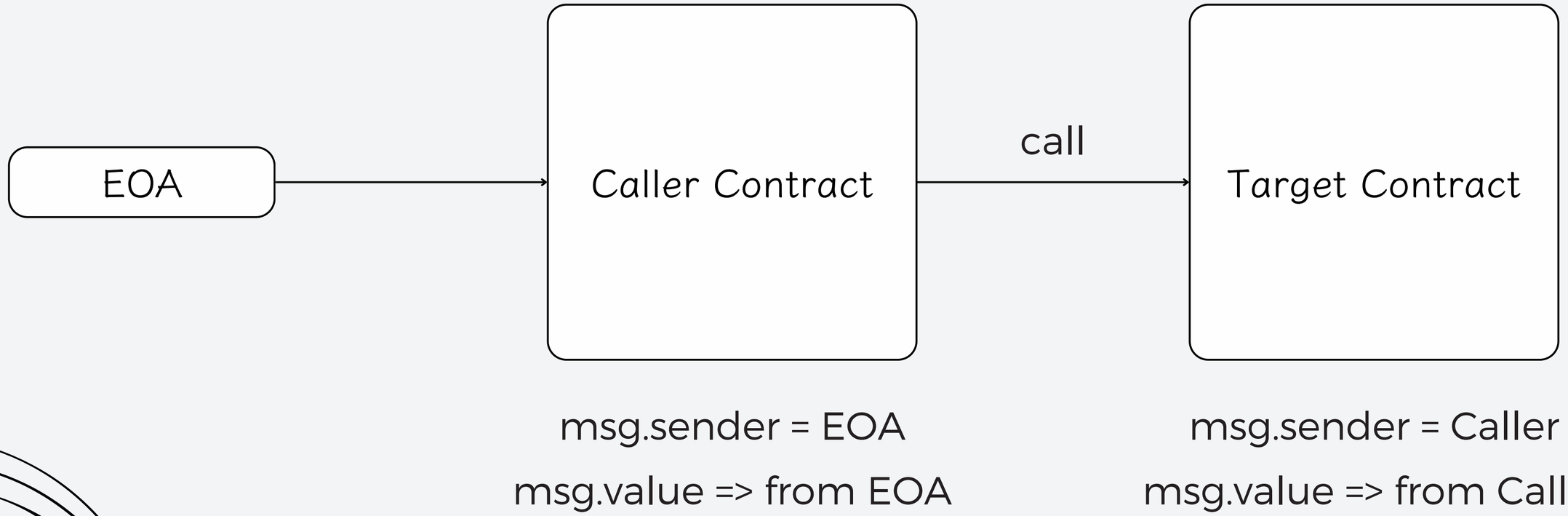
There is a function `add` in Counter
that will increase counter (slot0) value by 1

Call vs Delegatecall

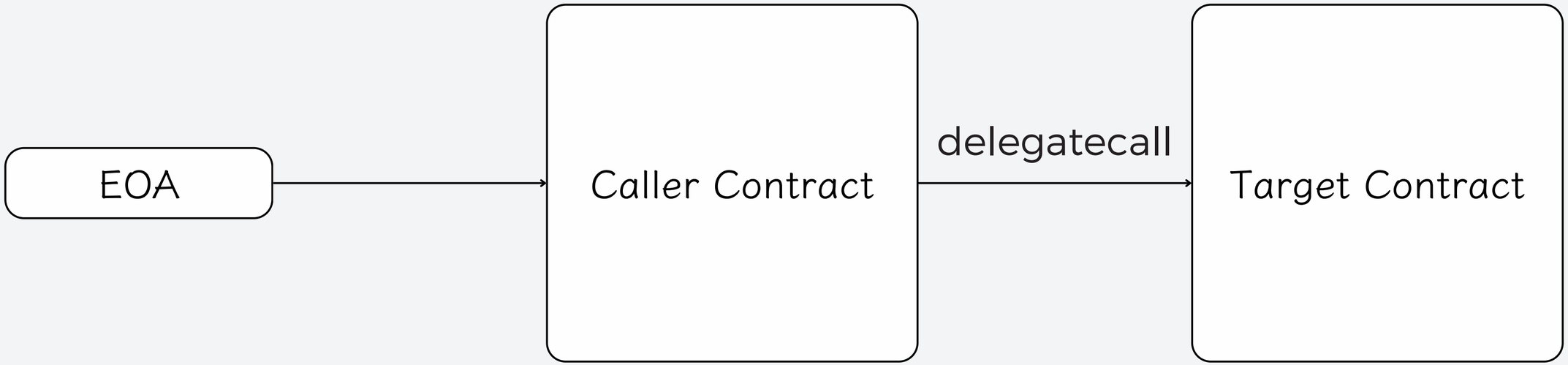


A delegatecall will execute the Callee function in the context of Caller contract

Call vs Delegatecall



Call vs Delegatecall



msg.sender = EOA

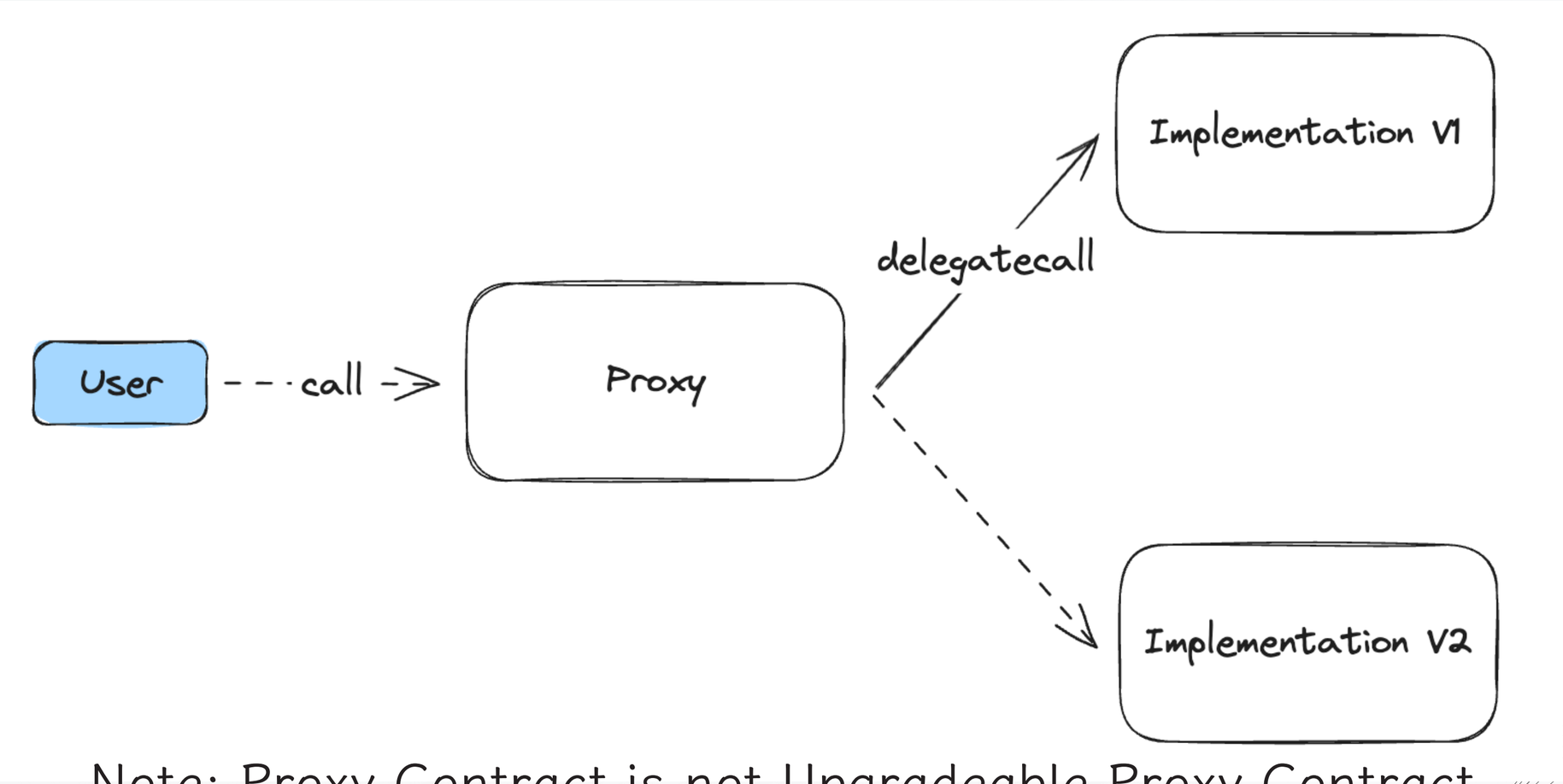
msg.value => from EOA

msg.sender = EOA

msg.value => from EOA

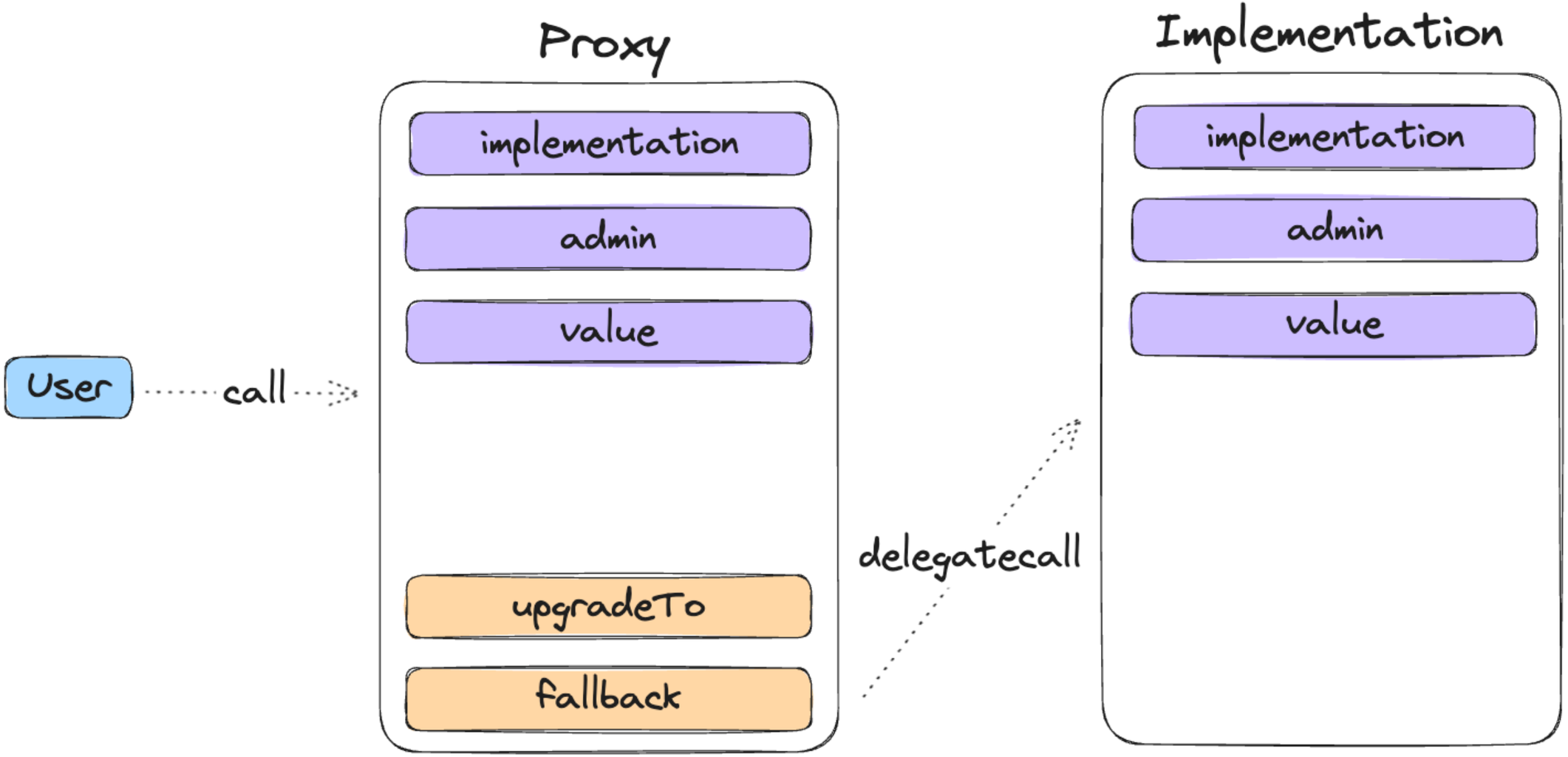
Upgradeable Proxy Pattern

Introduction to Upgradeable Proxy

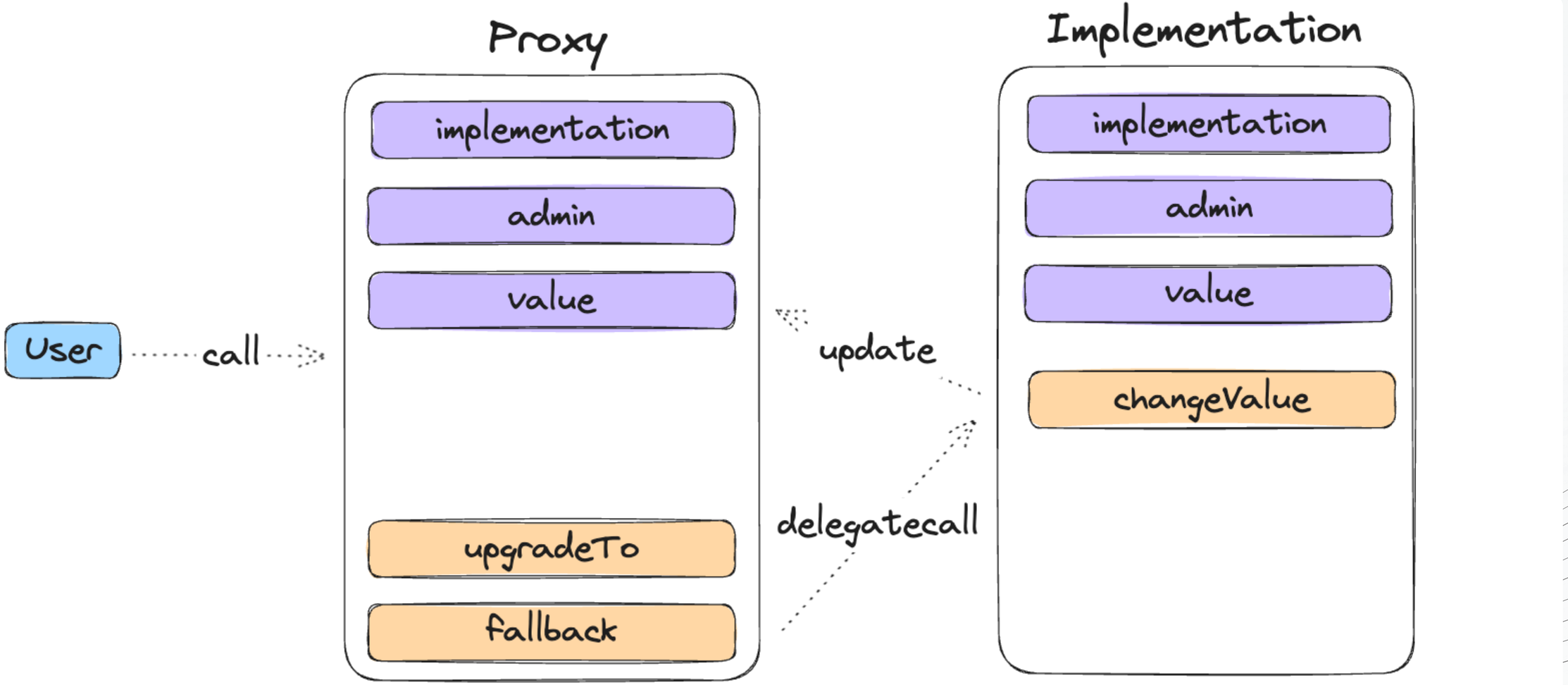


Note: Proxy Contract is not Upgradeable Proxy Contract

Introduction to Upgradeable Proxy



Introduction to Upgradeable Proxy



Initializable Proxy

The constructor in the implementation contract is unnecessary, as the state is updated in the proxy contract.

We replace the constructor with an `initialize` function for initialization. Since we expect the initialization to be done only once, access control is not necessary and should be handled atomically during the proxy upgrade.

Problem 1: The `initialize` function can be executed more than once.

Problem 2: The `initialize` function is not executed atomically in proxy upgrade.

Real World Attack - FLIX

Problem 1: The `initialize` function can be executed more than once.

```
/**
 * @dev constructor
 * @param initToken The token
 */
function init(
    IERC20 initToken,
    uint256 initPeriods,
    uint256 initInterval
) public {
    token = initToken;
    periods = initPeriods;
    interval = initInterval;
    owner = _msgSender();
}
```

The attacker execute this function and become the owner of the contract.

Loss	200 K
Chain	Ethereum
Transaction	0xbeef09ee9d694d2b24f3f367568cc6ba1dad591ea9f969c36e5b181fd301be82
PoC	Link

LAB 4

FLIX ATTACK POC

Bug Bounty - Arbitrum Nitro

Arbitrum Nitro

400 ETH Bounty

Upon initialization, the bridge address was set correctly, but the sequencerInbox remained uninitialized, leading to vulnerability;

This difference was explained by the postUpgradeInit function, which reset certain slots, leaving sequencerInbox and two booleans empty, despite the contract having been previously initialized.

Hackers in Arbitrum’s Inbox

This is the story of wiped storage slots and well-intentioned gas optimizations that led to a multi-million dollar vulnerability affecting...

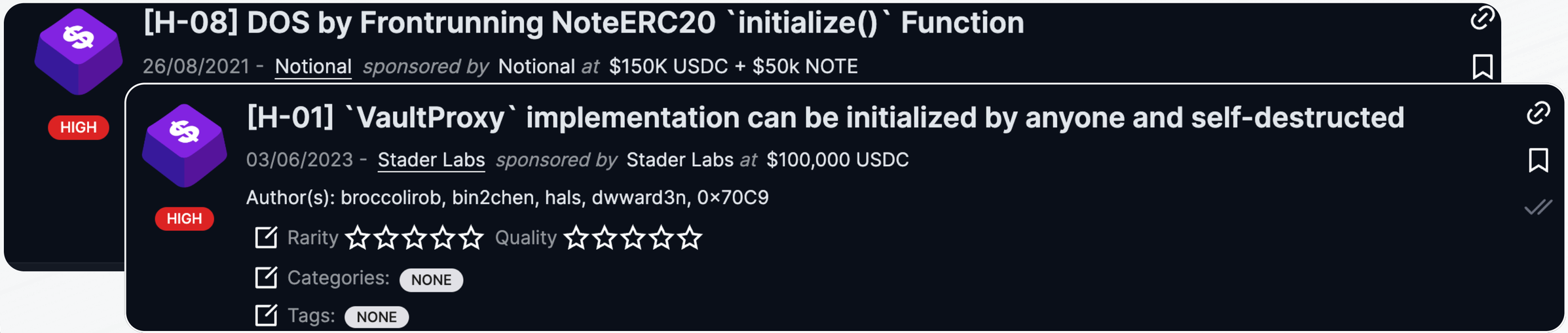
Medium / Sep 20, 2022

Audit Contests - Notional

Problem 2: the `initialize` is not executed atomically during the deployment phase.

Audit Link Severity HIGH

Front-run: when someone intercepts a pending transaction and places their own transaction ahead of it in the blockchain order, often to gain a financial advantage.

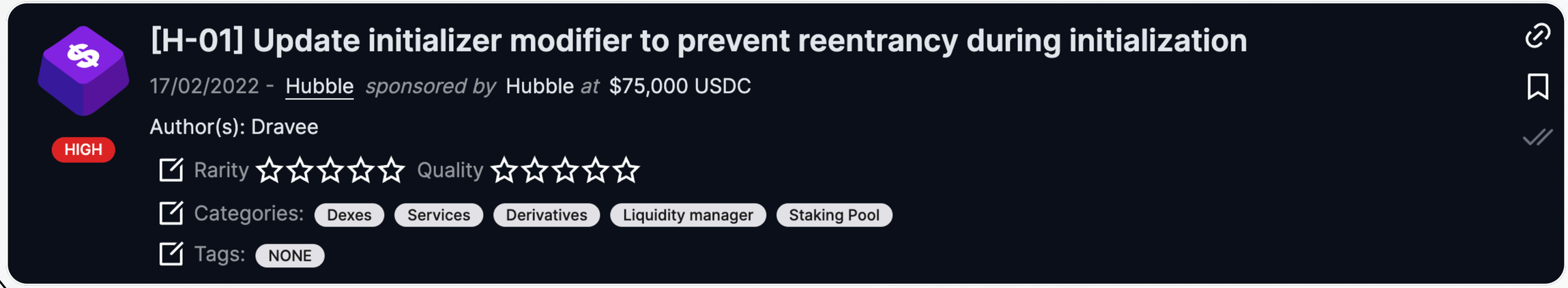


Solution

Add the `initializer` modifier to the `initialize` function. If the function has already been executed, it cannot be reinitialized.

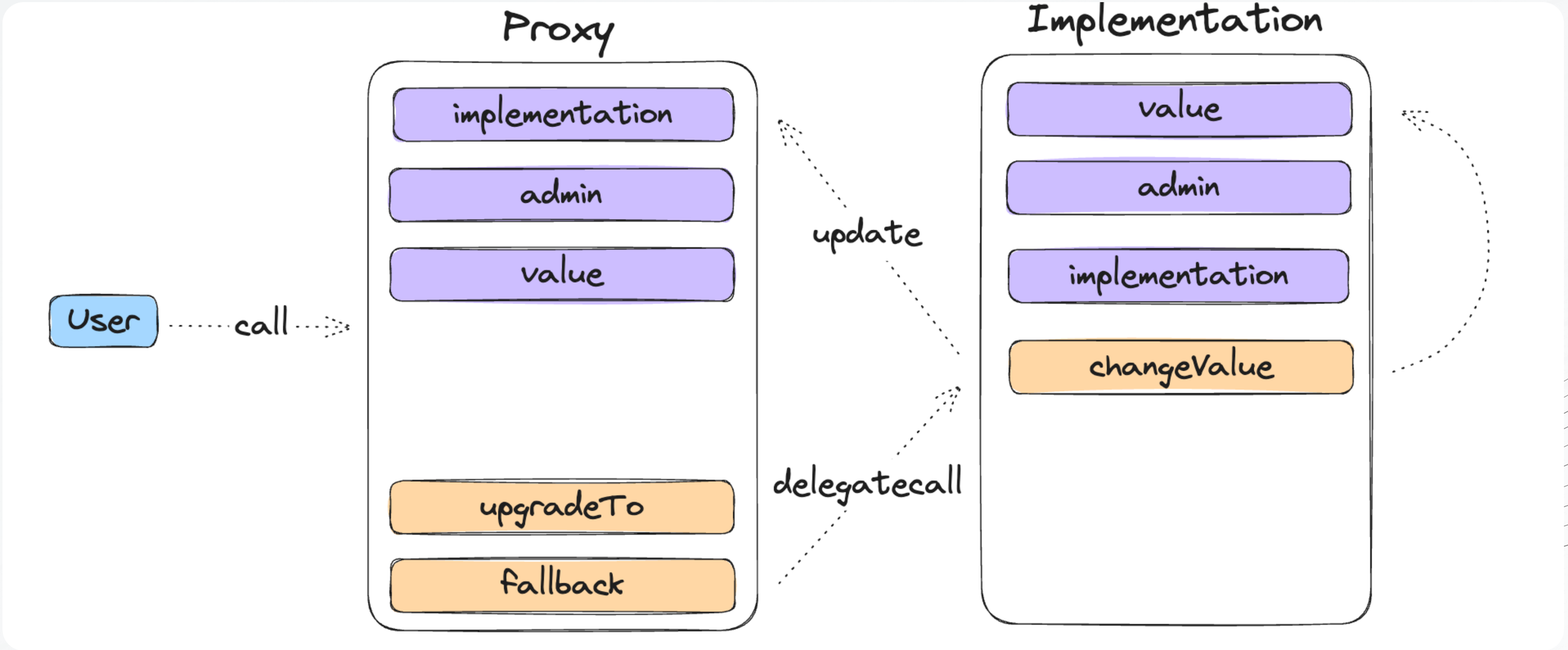
OpenZeppelin: The Initializable library offers related utility helpers.

NOTE: Check the OpenZeppelin version, as some have critical issues.



Storage Collision

We execute the callee contract's function in the context of the caller contract. What if the storage layouts are inconsistent?"



Real World Attack - Audius

Problem 1: The `initialize` function can be executed more than once.

Loss	1.08 M
Chain	Ethereum
Transaction	0xfefd829e246002a8fd061eede7501bccb6e244a9aacea0ebceaecef5d877a984
PoC	Link

- Call initialize function in three contracts - Governance, Staking, DelegateManagerV2
- Create malicious proposal and delegate voting power to the hacker itself.
- Pass the proposal and drain the token stored in the treasury.

Real World Attack - Audius

Proxy

forge inspect proxy.sol:AudiusAdminUpgradeabilityProxy storage --pretty

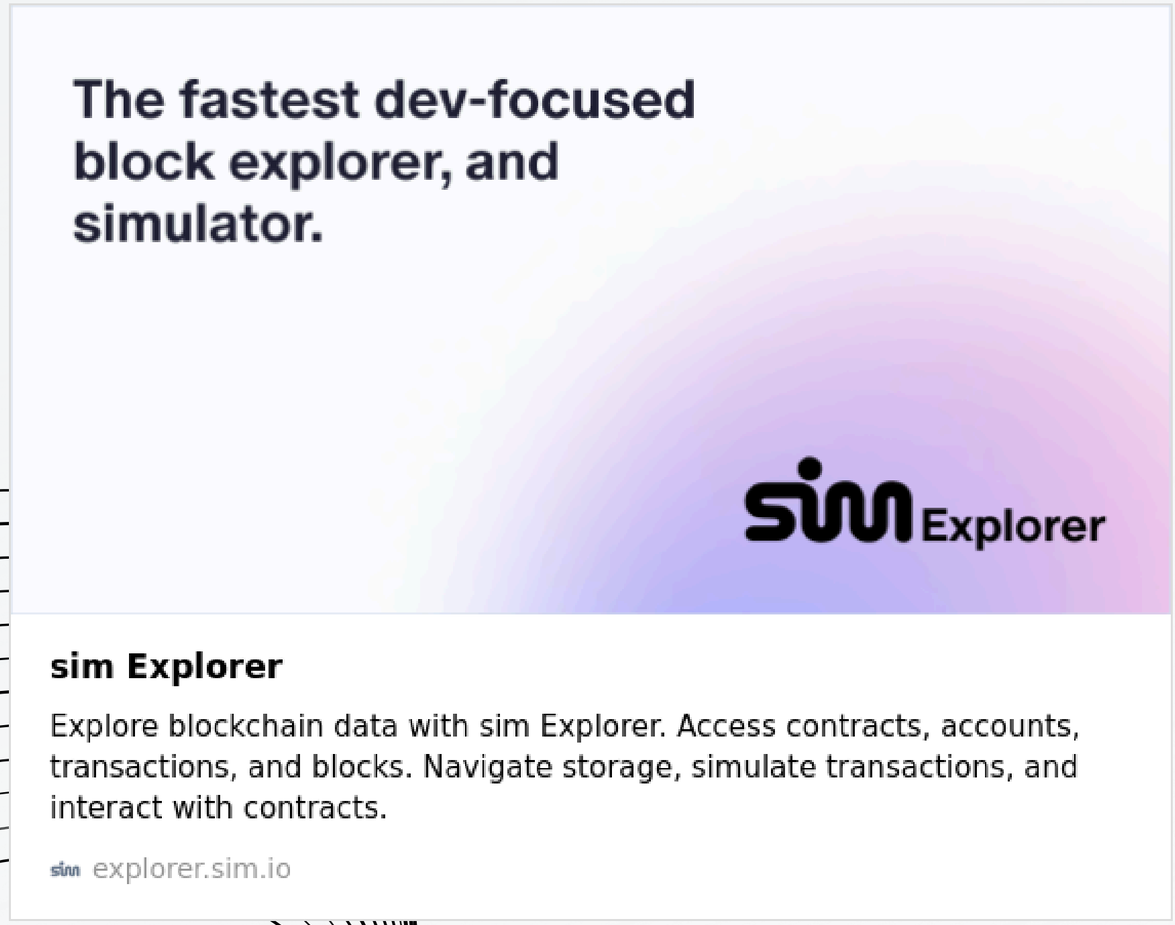
```
(venv) forge inspect proxy.sol:AudiusAdminUpgradeabilityProxy storagelayout --pretty
| Name       | Type    | Slot | Offset | Bytes | Contract                                          |
|------------|---------|------|--------|-------|---------------------------------------------------|
| proxyAdmin | address | 0    | 0      | 20    | src/proxy.sol:AudiusAdminUpgradeabilityProxy      |
```

Implementation

forge inspect inspect staking.sol:Staking storage --pretty

```
(venv) forge inspect staking.sol:Staking storagelayout --pretty
| Name                          | Type                                          | Slot | Offset | Bytes | Contract                  |
|-------------------------------|-----------------------------------------------|------|--------|-------|---------------------------|
| initialized                   | bool                                          | 0    | 0      | 1     | src/staking.sol:Staking   |
| initializing                  | bool                                          | 0    | 1      | 1     | src/staking.sol:Staking   |
| ______gap                     | uint256[50]                                   | 1    | 0      | 1600  | src/staking.sol:Staking   |
| isInitialized                 | bool                                          | 51   | 0      | 1     | src/staking.sol:Staking   |
| governanceAddress             | address                                       | 51   | 1      | 20    | src/staking.sol:Staking   |
| claimsManagerAddress          | address                                       | 52   | 0      | 20    | src/staking.sol:Staking   |
| delegateManagerAddress        | address                                       | 53   | 0      | 20    | src/staking.sol:Staking   |
| serviceProviderFactoryAddress | address                                       | 54   | 0      | 20    | src/staking.sol:Staking   |
| stakingToken                  | contract ERC20                                | 55   | 0      | 20    | src/staking.sol:Staking   |
| accounts                      | mapping(address => struct Staking.Account)    | 56   | 0      | 32    | src/staking.sol:Staking   |
| totalStakedHistory            | struct Checkpointing.History                  | 57   | 0      | 32    | src/staking.sol:Staking   |
```

Real World Attack - Audius



Visualization Tool

Proxy

0xe6d97b2099f142513be7a2a068be040656ae4591

Staking

0xea10fd3536fce6a5d40d55c790b96df33b26702f

Real World Attack - Audius

Proxy

0x4deca517d6817b6510798b7328f2314d3003abac proxyAdmin

Staking

initializing initialized

Real World Attack - Audius

Invalid Initializer, anyone can reinitialize the contract

```
modifier initializer() {
  require(initializing || isConstructor() || !initialized, "Contract instance has already been initialized");
```

```
function initialize(
    address _tokenAddress,
    address _governanceAddress
) public initializer
{
    require(Address.isContract(_tokenAddress), ERROR_TOKEN_NOT_CONTRACT);
    stakingToken = ERC20(_tokenAddress);
    _updateGgovernanceAddress(_governanceAddress);
    InitializableV2.initialize();
}
```

Allow attacker to update governance address.

Real World Attack - Audius

Invalid Initializer, anyone can reinitialize the contract

Initialize Staking Contract

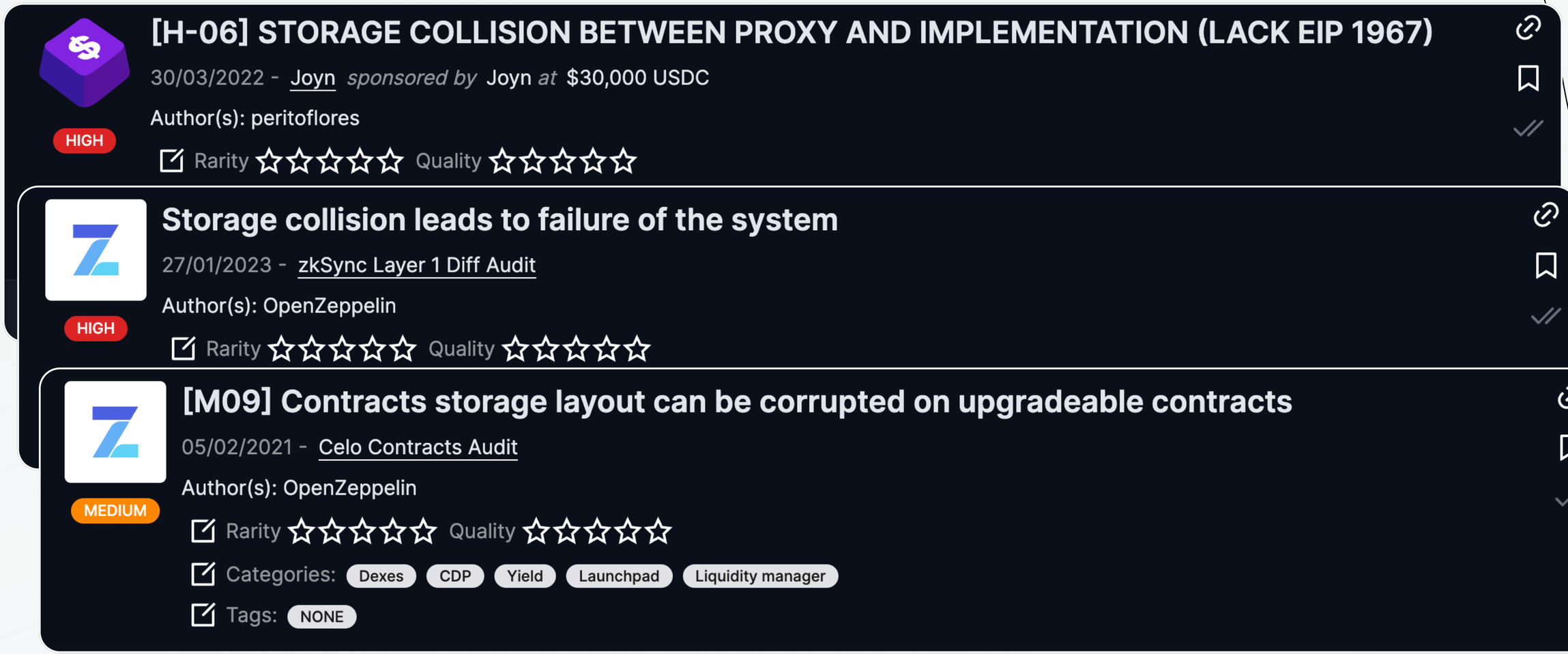
LAB 5

AUDIUS ATTACK POC

Storage Collision Examples

Protocol	Date	Chain	Loss (USD)
Pike Finance	2024-04-30	Ethereum	1.4 M
Telcoin	2023-12-25	Polygon	1.24 M
EFVault	2023-02-24	Polygon	5.1 M
Furucombo	2021-03-01	Ethereum	15 M

Storage Collision in Auditing Contests



Solution

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

Solution

ERC-1967: Proxy Storage Slots

Logic Contract

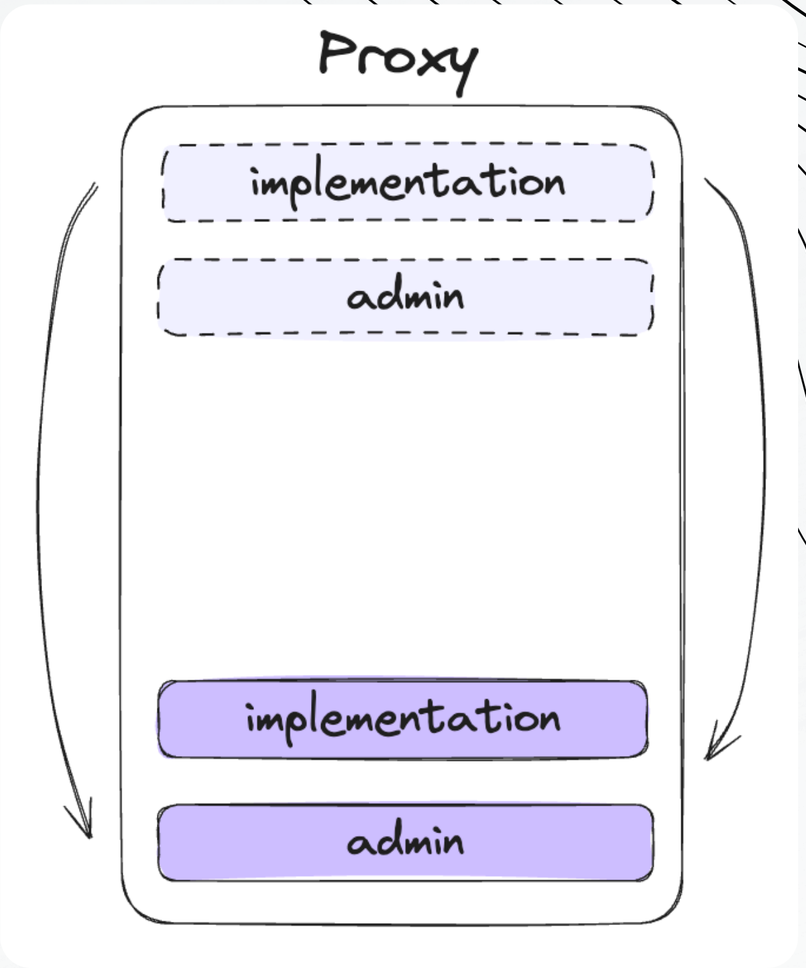
bytes32(uint256(keccak256('eip1967.proxy.implementation')) - 1)

0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc

Proxy Admin

bytes32(uint256(keccak256('eip1967.proxy.admin')) - 1))

0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103



Upgrade Pattern

Pattern In the upgradeable proxy pattern, there should have some upgrade logic.

Question Where to put the upgrade logic, there are two methods.

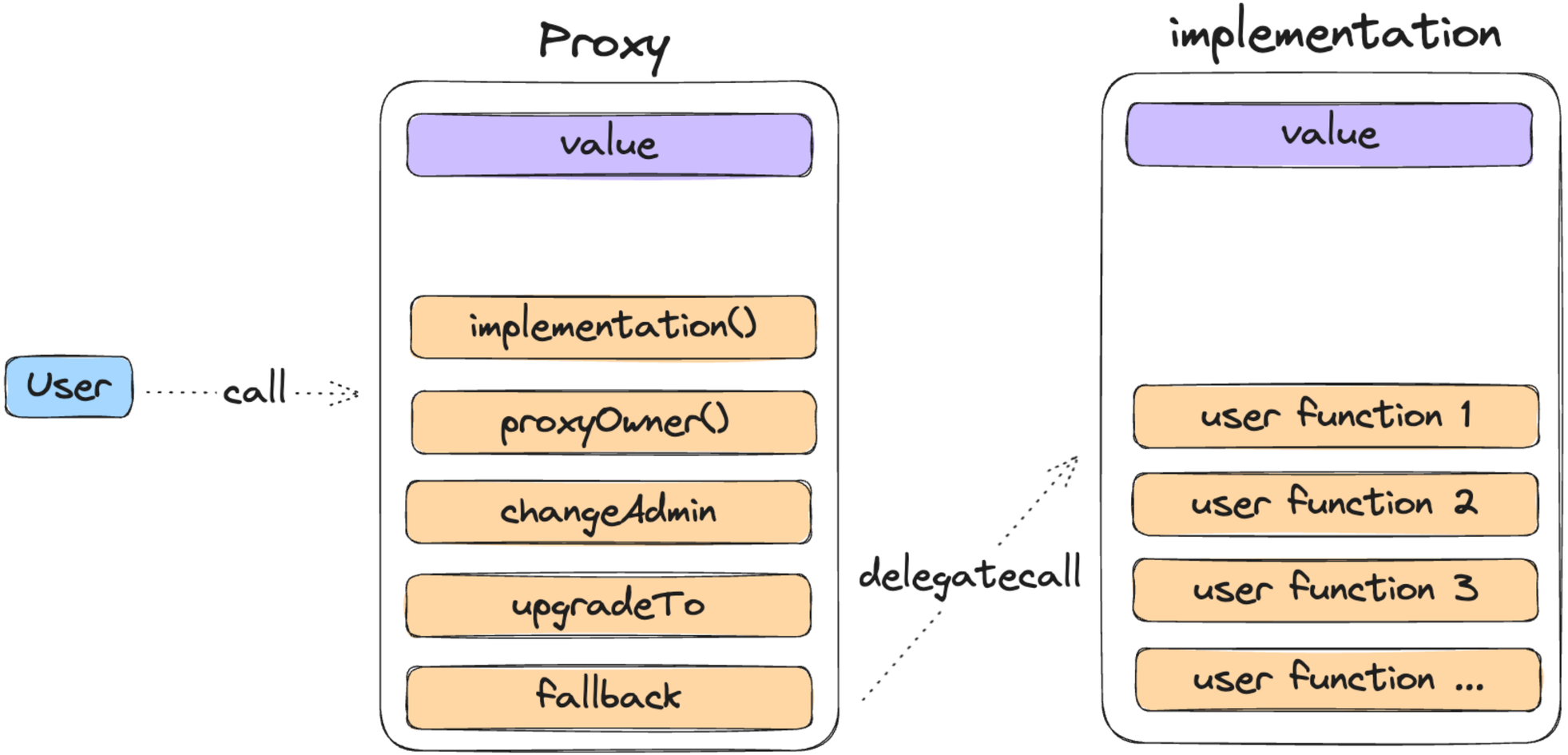
Transparent Proxy Pattern (**TPP**): the upgrade logic located in the proxy part

Universal Upgradeable Proxy Standard (**UUPS**): the upgrade logic located in the logic part

LAB 6

TPP & VULNERABILITY

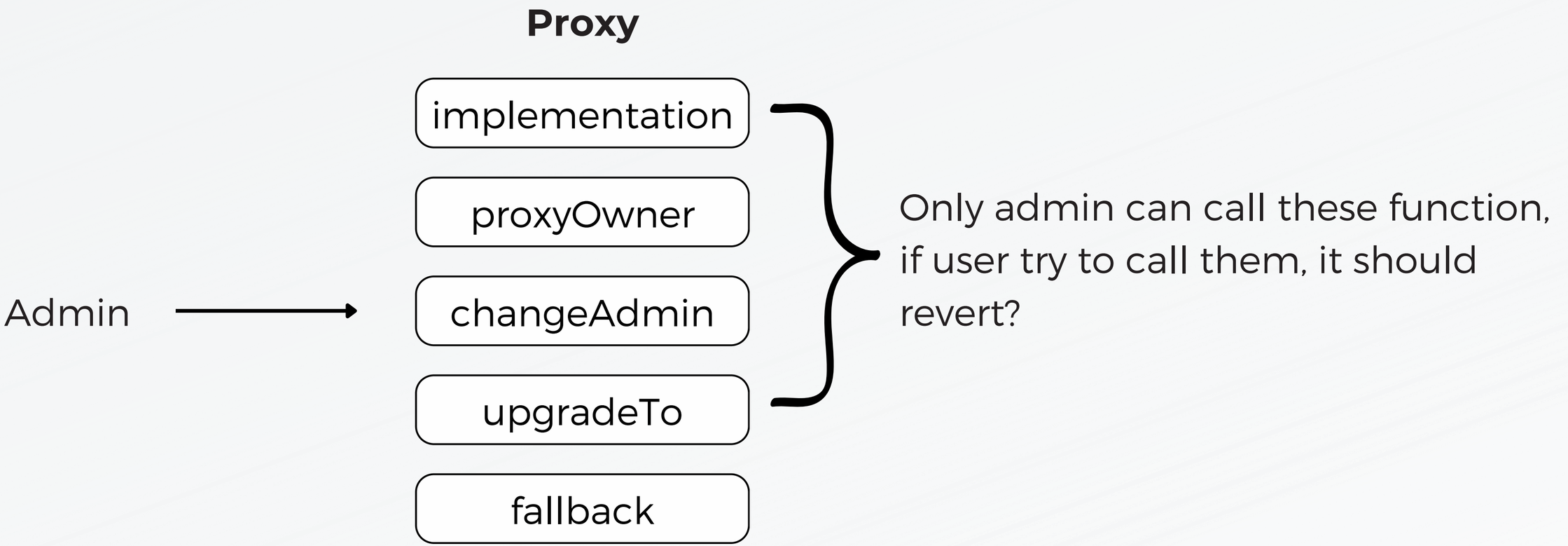
Transparent Proxy Pattern



* for simplicity, ignore implementation and admin variable here

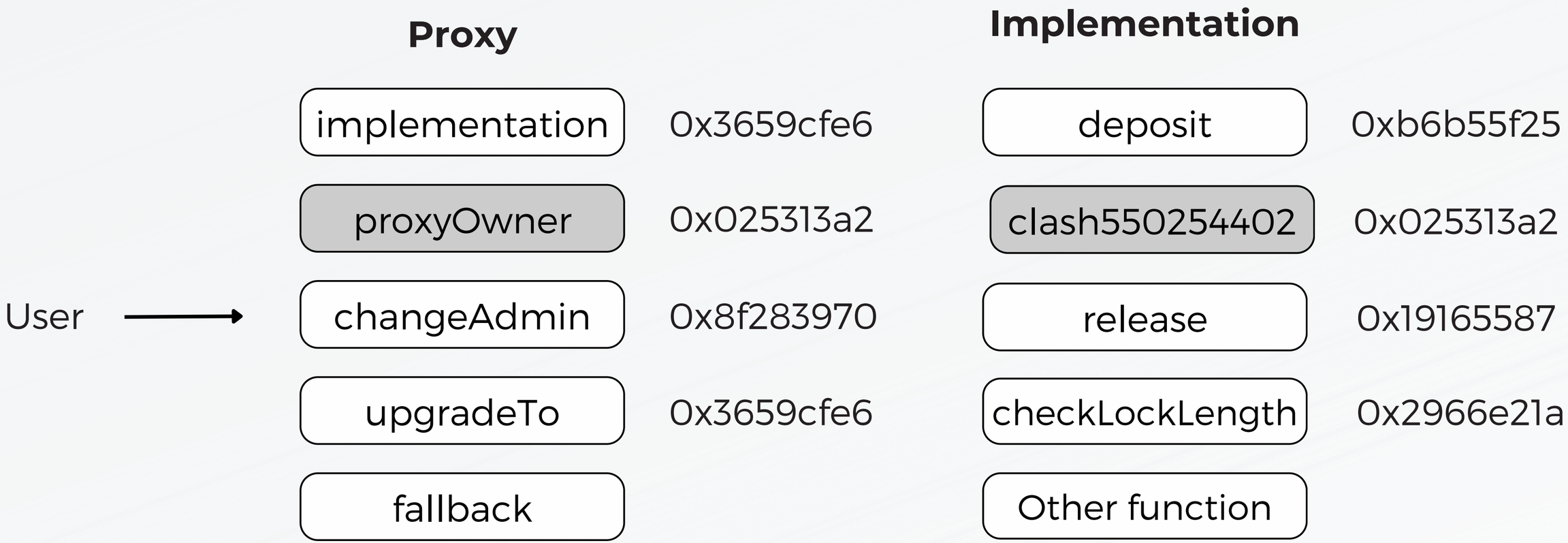
Transparent Proxy Pattern

If you are admin, privileged function will be executed, and the proxy will not delegate your calldata to implementation contract.



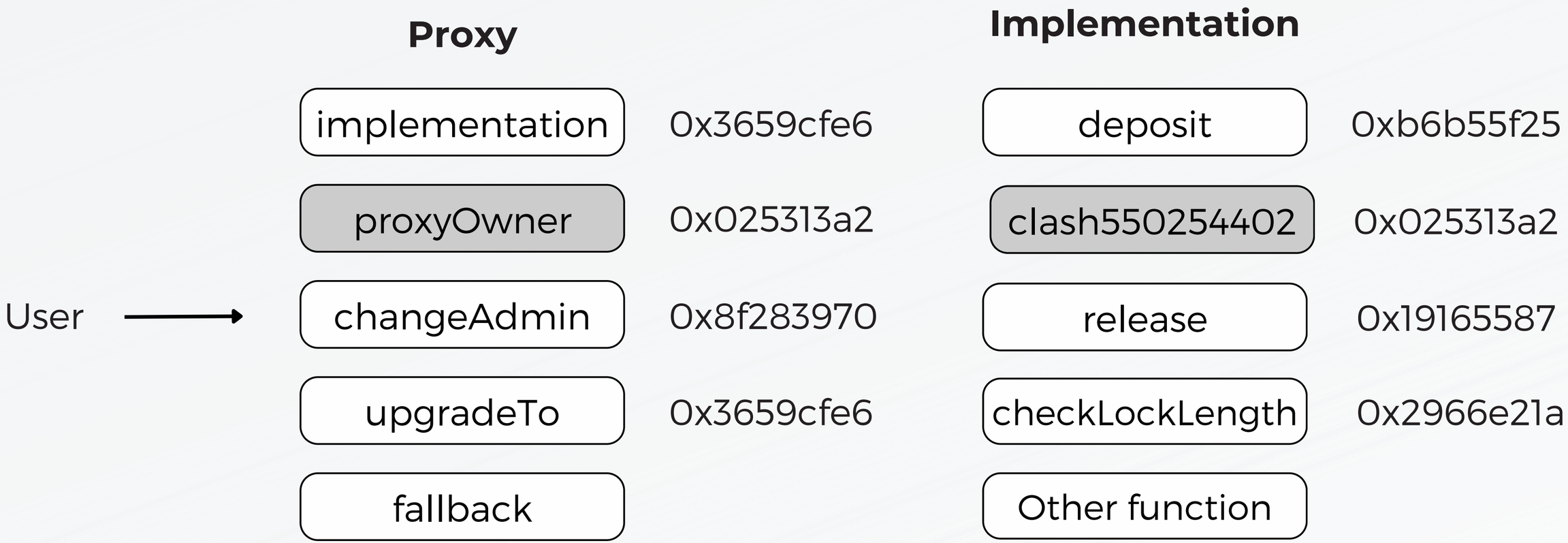
Transparent Proxy Pattern

If you are admin, privileged function will be executed, what if I always revert if normal users try to call them?



Transparent Proxy Pattern

The user function - clash550254402, will never be called since all the external call by users will always revert



Function Clashing

Recap: function selector = bytes4(keccak256(function signature))

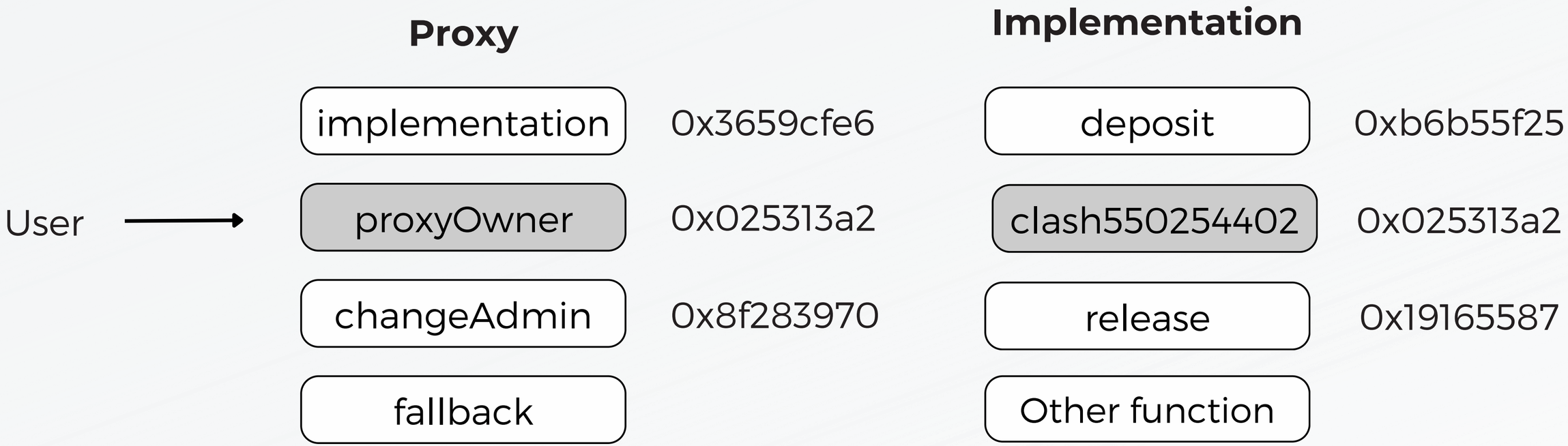
Number Of Combination = 16 ^ 8 = 2 ^ 32 = 4,294,967,296 -> It is possible for collision

function selector	function signature 1	function signature 1
0x025313a2	proxyOwner()	clash550254402()
0x3659cfe6	upgradeTo(address)	upgradeTo_790AA3D()
0x42966c68	burn(uint256)	collate_propagate_storage(bytes16)

Transparent Proxy Pattern

Solution 1: If the caller is not admin, forward the calldate through delegatecall

Solution 2: Place all functions in the implementation contract -> UUPS



LAB 7

UUPS & VULNERABILITY

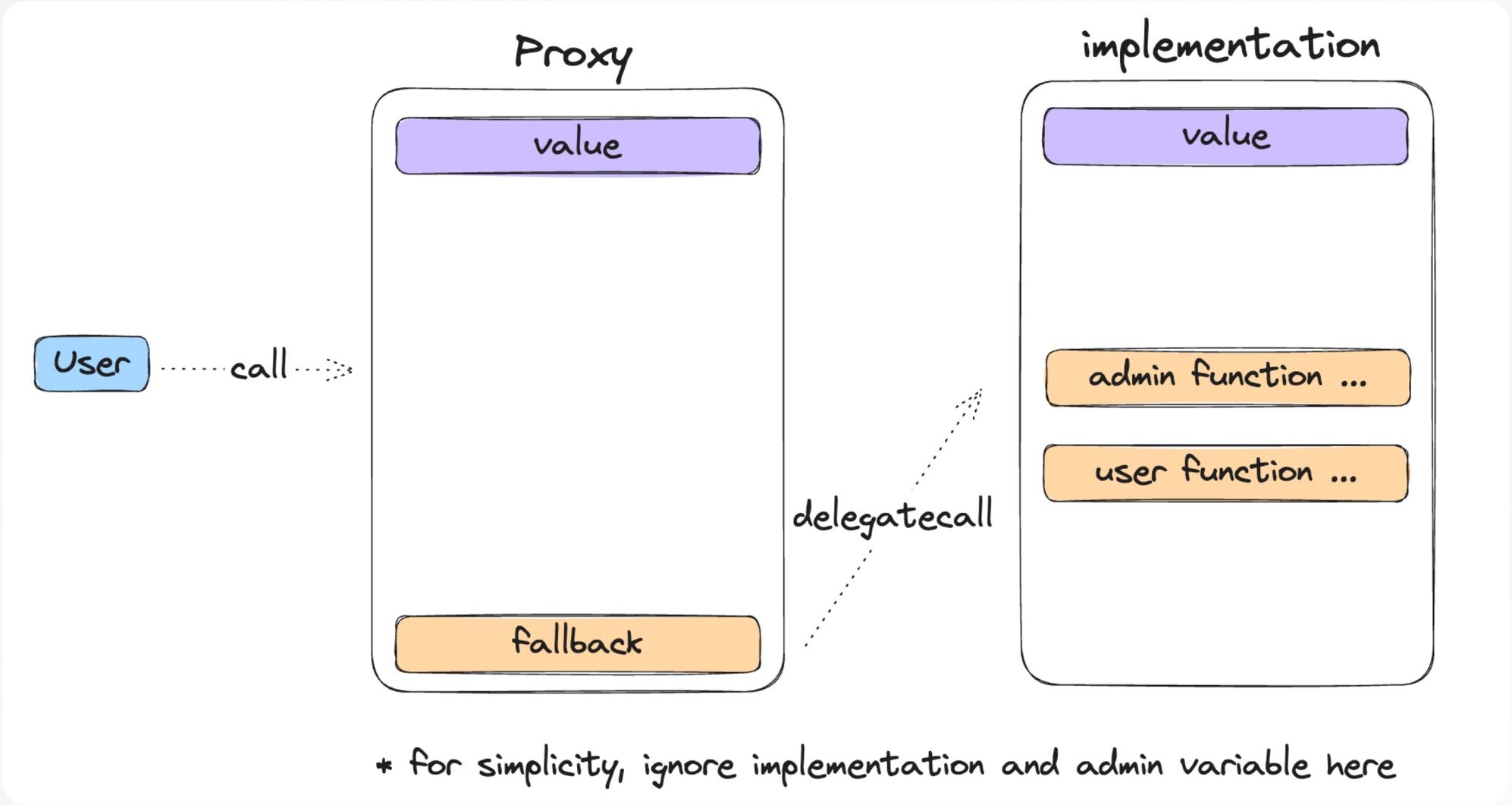
Universal Upgradeable Proxy Standard

ERC-1822: Universal Upgradeable Proxy Standard (UUPS)

The following describes a standard for proxy contracts which is universally compatible with all contracts, and does not create incompatibility between the proxy and business-logic contracts. This is...

Ethereum Improvement Proposals

Universal Upgradeable Proxy Standard



UUPS Vulnerability

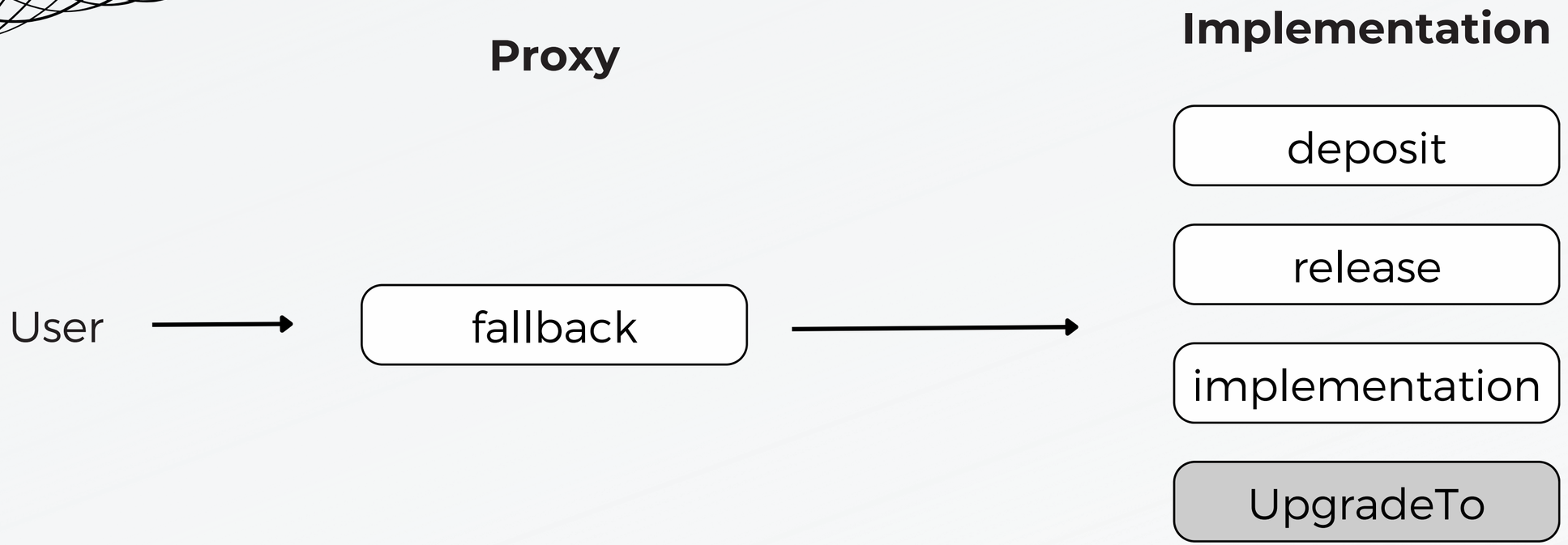
UUPSUpgradeable vulnerability in OpenZeppelin Contracts

Critical **frangio** published **GHSA-5vp3-v4hc-gx76** on Sep 15, 2021

Package	Affected versions	Patched versions
@openzeppelin/contracts (npm)	>= 4.1.0 < 4.3.2	4.3.2
@openzeppelin/contracts-upgradeable (npm)	>= 4.1.0 < 4.3.2	4.3.2

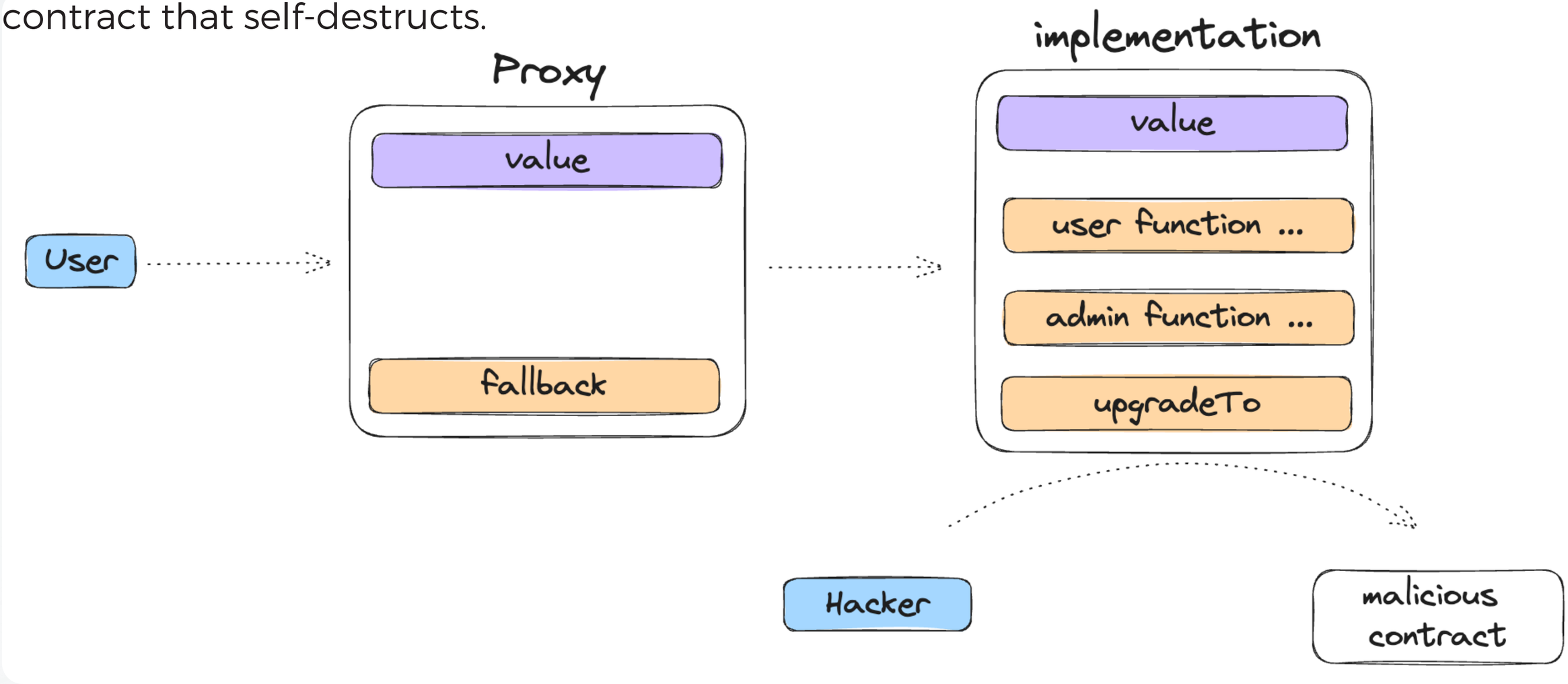
Please do not use this version of OppenZeppelin upgradeable proxy template

UUPS Vulnerability



UUPS Vulnerability

Since the upgrade logic is located in the implementation contract, upgrade it to a contract that self-destructs.



selfdestruct

pre-cancun

Destroy the specified contract bytecode, ensuring the transfer of the current balance to the designated account, which always succeeds, even though there is no receive or fallback function in the contract account.

cancun

Send all Ether from the account to the target; if the selfdestruct instruction is executed within the same transaction as contract creation, the contract can be destroyed at the end of the transaction.

Bug Bounty & Asset Rescue

Protocol		Root Cause
KeeperDAO	44 M TVL	The uninitialized logic contract behind a UUPS proxy, allowing arbitrary delegatecalls and selfdestruct instructions that could disable the proxy and make funds inaccessible.
Rivermen NFT	6.95 M TVL	
Wormhole	10 M Bounty	The uninitialized UUPS implementation contract, enabling an attacker to use a forged Guardian set to contract upgrade and destroy it with SELFDESTRUCT
polygon	NaN	StakeManagerProxy and StakeManager could allow an attacker to deny access to the contracts.

Bug Bounty & Asset Rescue

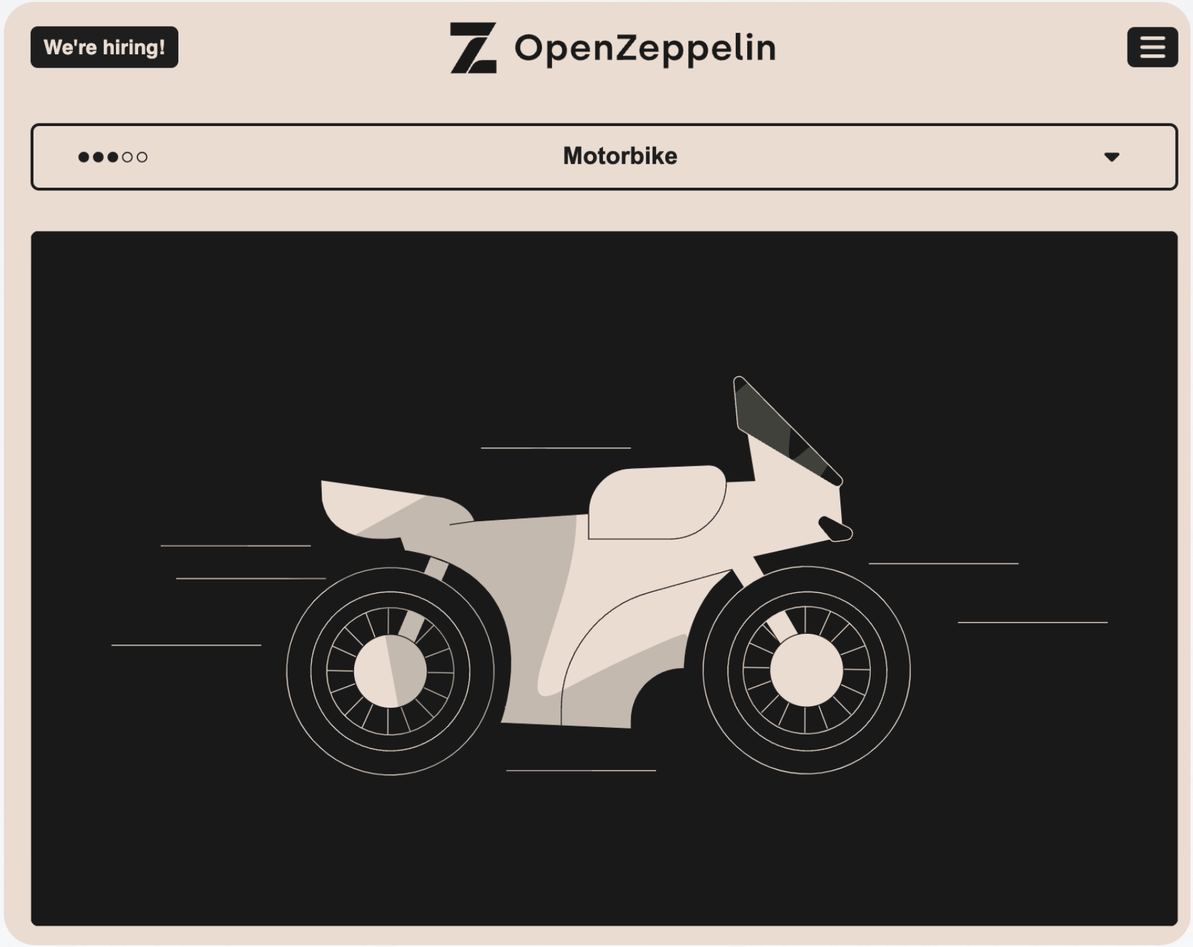
Protocol		Root Cause
Aave	25 K bounty	a misconfigured logic contract let anyone initialize the LendingPool, set a malicious _addressesProvider, and use liquidationCall to selfdestruct the implementation via DELEGATECALL.
Agave	20 K Bounty	
Harvest	200 K bounty	an attacker can exploit contract by initializing it, setting another contract that calls SELFDESTRUCT, and invoking doHardWork().

Bug Bounty & Asset Rescue

The OpenZeppelin team searches all the contracts that utilize this vulnerable version of the template and helps them initialize the variable for mitigation.

Before sharing the advisory, we sent transactions to initialize more than 150 uninitialized UUPS implementation contracts across Ethereum Mainnet, Polygon, xDAI, Binance, and Avalanche. We found no instances on Fuse, Fantom, Arbitrum, or Optimism. These transactions were all sent from EOA `0x37E8d216c3f6c79eC695FBD0cB9842e62fB84370`, and through a batching contract at `0x310fAC62C976d8F6FDFA34332a56EA1a05493b5b`. We relayed the transaction on mainnet privately using Taichi 85 to protect against generalized frontrunning bots.

Ethernaut Challenge: Motorbike



Ethernaut is a CTF challenge provided by OpenZeppelin

Please check the simplified version in the following path.

Please check the simplified version in the following path.

forge test --match test <test-name>

Upgradeable Proxy Security Summary

Upgradeable Proxy Audit Tips

Audit Checklist for Upgradeable Proxy by Solodit

- [ ] Proxy/Upgradable
  - [ ] Is there a constructor in the proxied contract?
  - [ ] Is the `initializer` modifier applied to the `initialization()` function?
  - [ ] Is the upgradable version used for initialization?
  - [ ] Is the `authorizeUpgrade()` function properly secured in a UUPS setup?
  - [ ] Is the contract initialized?
  - [ ] Are `selfdestruct` and `delegatecall` used within the implementation contracts?
  - [ ] Are values in immutable variables preserved between upgrades?
  - [ ] Has the contract inherited the correct branch of OpenZeppelin library?
  - [ ] Could an upgrade of the contract result in storage collision?
  - [ ] Are the order and types of storage variables consistent between upgrades?

Upgradeable Proxy Audit Tips

Audit Checklist for Upgradeable Proxy by Solodit

Tip 1: Are the storage layout and data types consistent across upgrades to prevent potential storage collisions?

Tip 2: Are there potential issues with arbitrary calls or delegatecalls in the proxy pattern? Are selfdestruct and delegatecall utilized in implementation contracts?

Tip 3: Are immutable and constant variables preserved in proxies, potentially leading to inconsistency issues during upgrades?

Upgradeable Proxy Audit Tips

Audit Checklist for Upgradeable Proxy by Solodit

Tip 4 : Is there any initializer modifier for `initialize` function to prevent re-initialization?

Tip 5: Is necessary initialization logic located in `initialize` function instead of constructor in the implementation contract?

Tip 6: Is the contract being initialized during the deployment phase?

Upgradeable Proxy Audit Tips

Audit Checklist for Upgradeable Proxy by Solodit

Tip 7: Does initialization of the implementation contract which involves upgrade logic prevent malicious upgrades?

Tip 8: Is the proxy inherited from the appropriate OZ library to ensure there are no issues during contract initialization?

Tip 9: Are proper access controls implemented for the `authorizeUpgrade()` function to prevent unauthorized upgrades?

Oppenzeppelin Integration

Audit Checklist for Upgradeable Proxy by Solodit

Tip 7: Does initialization of the implementation contract which involves upgrade logic prevent malicious upgrades?

Tip 8: Is the proxy inherited from the appropriate OZ library to ensure there are no issues during contract initialization?

Tip 9: Are proper access controls implemented for the `authorizeUpgrade()` function to prevent unauthorized upgrades?

Reference

- Breaking Aave Upgradeability
- Wormhole Uninitialized Proxy Bugfix Review
- Hackers in Arbitrum’s Inbox
- Harvest Finance Uninitialized Proxies Bugfix Review— $200k Bounty
- Teller Bugfix Review and Bug Bounty Launch
- Forked protocols are not battle-tested: Agave Uninitialized Proxy Vulnerability
- Audius Governance Takeover Post-Mortem
- Furucombo Post-Mortem March 2021
- yAcademy Proxies Research
- OpenZeppelin - Proxies
- ERC-1967: Proxy Storage Slots
- ERC-1822: Universal Upgradeable Proxy Standard (UUPS)
- DeFiHackLabs
- Solodit

More Examples

Example Contract

- Link：https://github.com/Cyfrin/foundry-upgrades-cu
- Link：https://solidity-by-example.org/app/upgradeable-proxy/

Thank You For Listening