



# Upgradeable smart contracts security

Arseniy Reutov

**CTO Decurity.io**

# Agenda

- Why proxies?
- Upgradeability patterns
- Proxy storage collision
- Cases: OpenZeppelin, Wormhole, Audius
- Tools & techniques

# Why proxies?

# Smart contracts are immutable

Cons

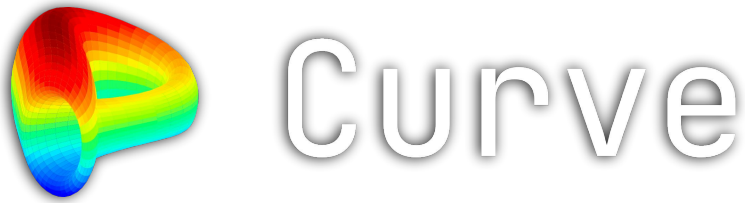
- Requires software quality of a Mars rover
- No way to fix bugs without redeploying a contract to a new address
- A single bug can be a disaster

Pros

- Can’t rug

# Immutable contracts examples





# Security Ops

- Find out normal parameters (minimum amount of liquidity, solvency criteria, price within specific range)
- Monitor (e.g. with Forta)
- React (pause the contract, remove liquidity, emergency exit)
- Patch

# Patching

- Why can’t we just deploy a new contract?
- Because DeFi is composable
- DeFi is not used only via a frontend, but by other contracts too
- If contract’s address changes you have to change it everywhere
- Some workarounds exist though: registry contracts and ENS resolution
- But most common practice: proxies

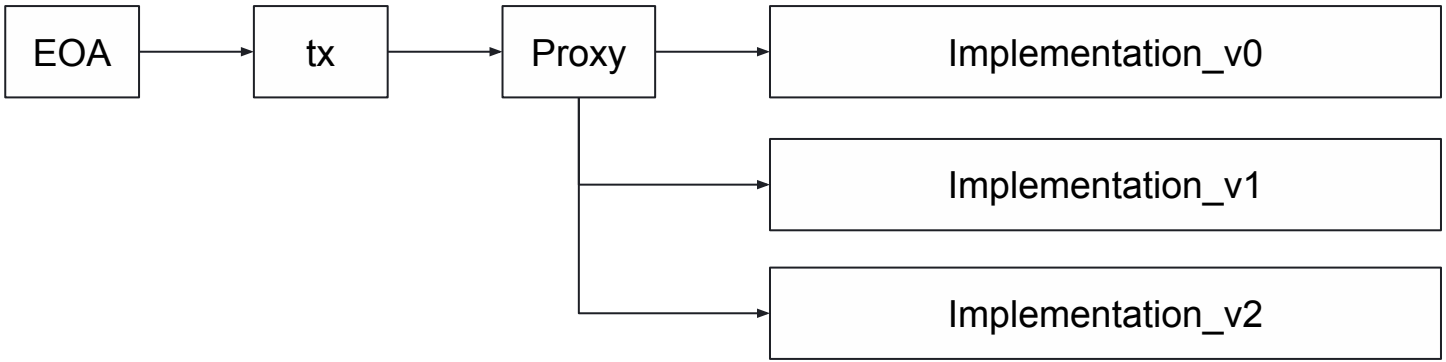


# Upgradeability patterns

# Upgrading via proxy

- Proxy contract is a wrapper
- Think of a reverse proxy in front of a web server
- The main function of a proxy: forward calls to the implementation contract
- The main property of a proxy: static address

# Upgrading via proxy




https://docs.openzeppelin.com/upgrades-plugins/1.x/proxies

# How is it achieved?



💡 delegatecall inside a fallback function

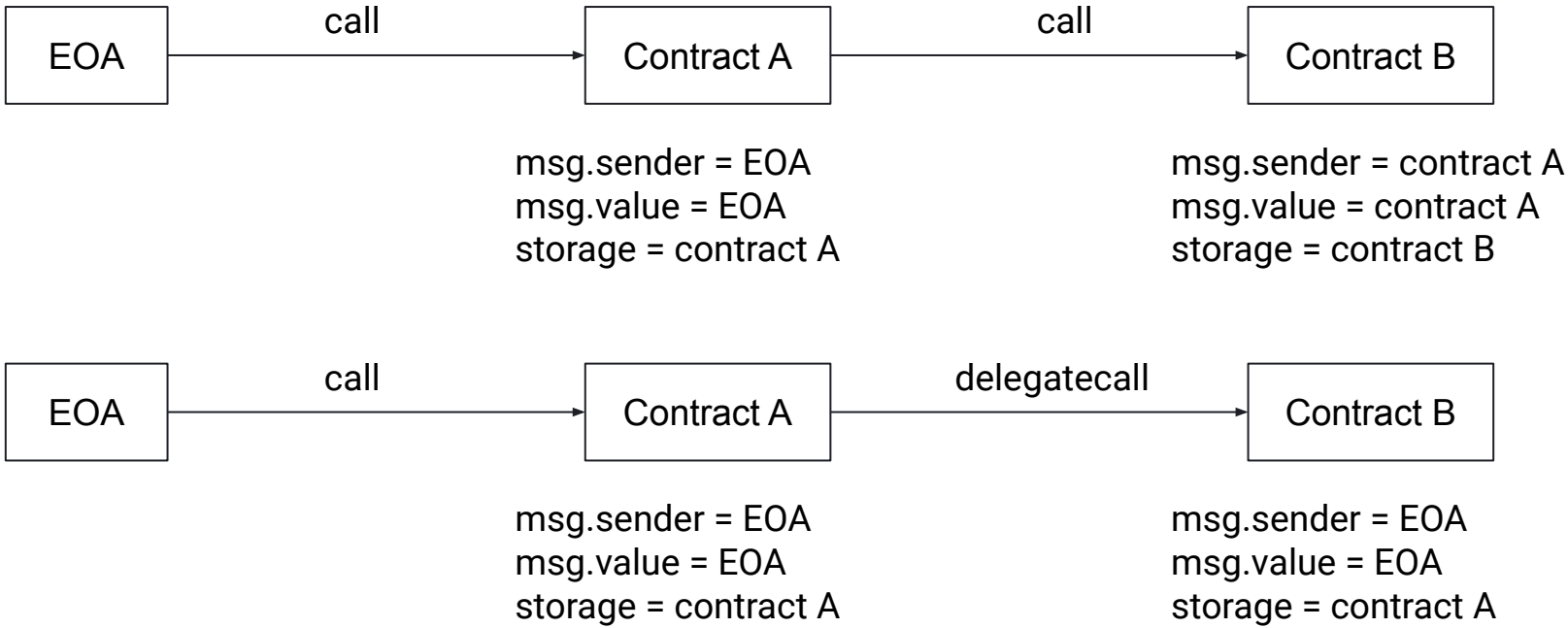
```
fallback() external payable {
    if (gasleft() <= 2300) {
        revert();
    }

    address target_ = target;
    bytes memory data = msg.data;
    assembly {
        let result := delegatecall(gas(), target_, add(data, 0x20), mload(data), 0, 0)
        let size := returndatasize()
        let ptr := mload(0x40)
        returndatacopy(ptr, 0, size)
        switch result
        case 0 { revert(ptr, size) }
        default { return(ptr, size) }
    }
}
```

# delegatecall

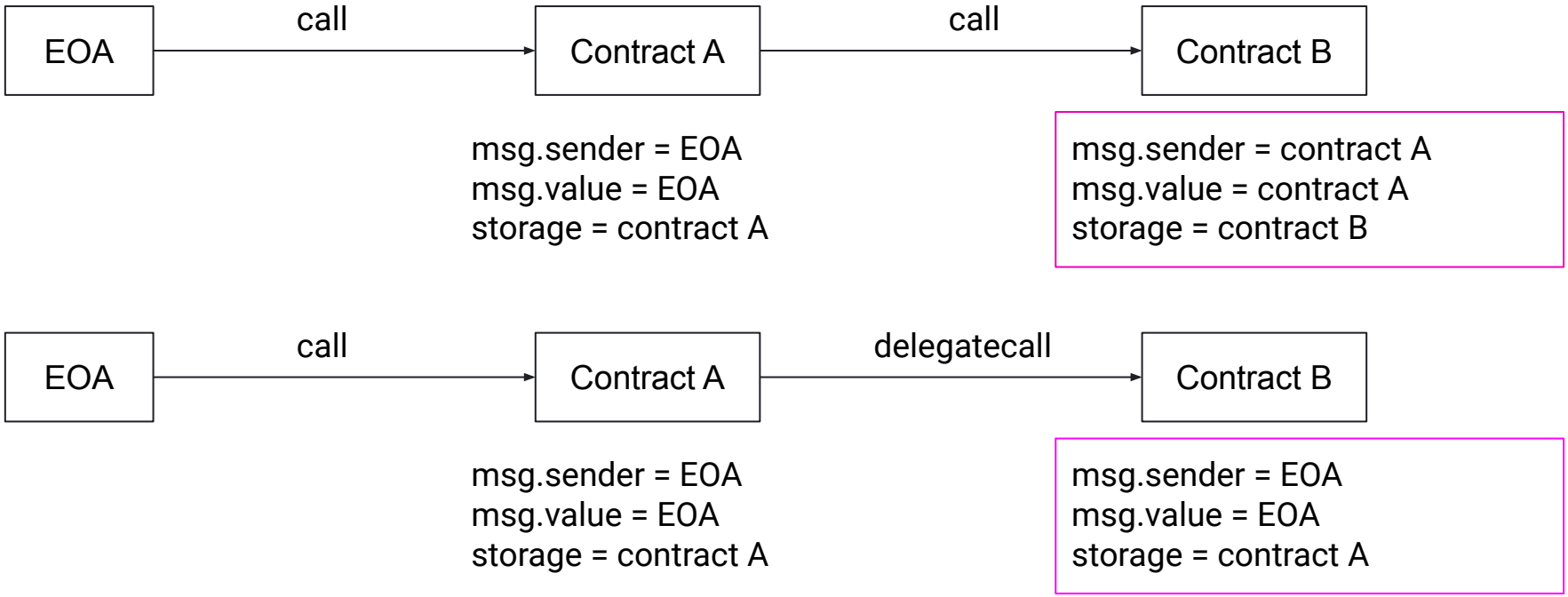
In EVM there are three ways of calling a function:

1. call - state mutable call, i.e. write
2. staticcall - non mutable call, i.e. read
3. delegatecall - mutable call, but on our own storage

# delegatecall vs call



# delegatecall vs call



# Proxy initialization

- Constructor is automatically called during contract deployment
- But this is no longer possible with proxies
- Because the constructor will change only the implementation contract's storage
- Solution – change the constructor to a regular function
- Usually this function is called `initialize()`
- It has `initializer` modifier which prevents re-initialization

# Proxy patterns

1. Transparent proxy pattern (TPP)
2. Universal upgradeable proxy system (UUPS)

Difference is that TPP proxy contains upgrade logic, while UUPS off-loads this logic to the implementation contract.

| msg.sender | owner() | upgradeTo() | transfer() |
|---|---|---|---|
| Admin | returns proxy owner | upgrades proxy | reverts |
| Other account | returns ERC20 owner | reverts | sends ERC20 transfer |

Credit: @OpenZeppelin

# Storage layouts

Proxy has to store at least one variable, which is the implementation address.

There are two storage layouts:

1. Structured storage - usually achieved by inheriting the same contract by both proxy and implementation
2. Unstructured storage - implementation address is stored in a pseudo-random slot location, such that an overwrite possibility is tiny (EIP-1967)

# Proxy storage collisions

# EVM Storage

- EVM storage is a sequence of 32-byte slots, max length is 2**256
- There is no allocator, contract can read & write everywhere

| | |
|---|---|
| slot 0 | uint256 foo |
| slot 1 | uint256 bar |
| slot 2 | items.length=2 |
| slot 3 | |
| slot keccak256(2) | items[0]=12 |
| slot keccak256(2)+1 | items[1]=42 |

```
uint256 foo;
uint256 bar;
uint256[] items;

function allocate() public {
    require(0 == items.length);

    items.length = 2;
    items[0] = 12;
    items[1] = 42;
}
```

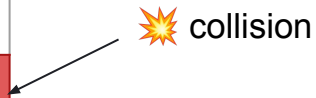
https://mixbytes.io/blog/collisions-solidity-storage-layouts

# Structured storage

| Proxy | Implementation |
| --- | --- |
| address _implementation | address _owner |
| ... | mapping _balances |
| ... | uint256 _supply |
| ... | ... |

# Structured storage

| Proxy | Implementation |
| --- | --- |
| address _implementation | address _owner |
| ... | mapping _balances |
| ... | uint256 _supply |
| ... | ... |



# Unstructured storage

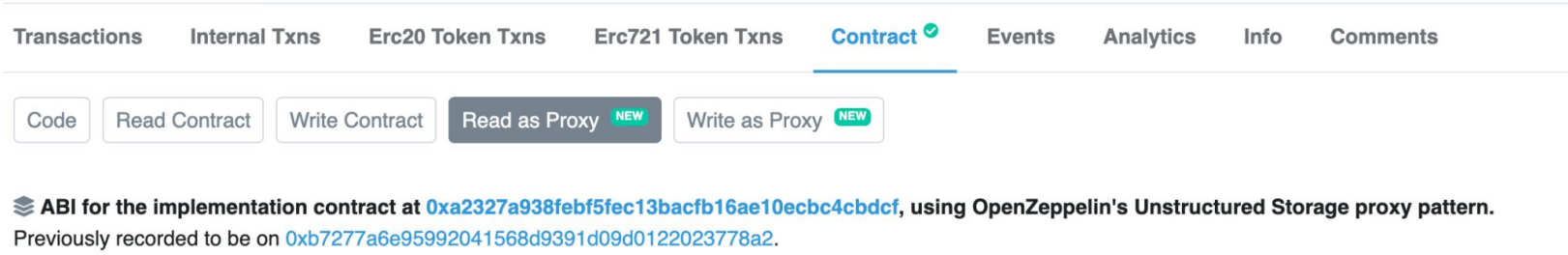
| Proxy | Implementation |
| --- | --- |
| ... | address _owner |
| ... | mapping _balances |
| ... | uint256 _supply |
| ... | ... |
| ... | |
| ... | |
| address _implementation | |
| ... | |

# Unstructured storage

| Proxy | Implementation |
| --- | --- |
| ... | address _owner |
| ... | mapping _balances |
| ... | uint256 _supply |
| ... | ... |
| ... | |
| ... | |
| address _implementation | |
| ... | |

# EIP-1967

```
bytes32 private constant implementationPosition = bytes32(uint256(
    keccak256('eip1967.proxy.implementation')) - 1
));
```

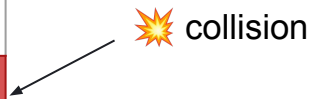


# Storage collisions between implementations

| Implementation_v0 | Implementation_v1 |
| --- | --- |
| address _owner | address _lastContributor |
| mapping _balances | address _owner |
| uint256 _supply | mapping _balances |
| ... | uint256 _supply |

# Storage collisions between implementations

| Implementation_v0 | Implementation_v1 |
| --- | --- |
| address _owner | address _lastContributor |
| mapping _balances | address _owner |
| uint256 _supply | mapping _balances |
| ... | uint256 _supply |



# Storage collisions between implementations

| Implementation_v0 | Implementation_v1 |
| --- | --- |
| address _owner | address _owner |
| mapping _balances | mapping _balances |
| uint256 _supply | uint256 _supply |
| ... | address _lastContributor |

# Storage collisions between implementations

| Implementation_v0 | Implementation_v1 |
| --- | --- |
| address _owner | address _owner |
| mapping _balances | mapping _balances |
| uint256 _supply | uint256 _supply |
| ... | address _lastContributor |


```
/**
 * @dev This empty reserved space is put in place to allow future versions to add new
 * variables without shifting down storage in the inheritance chain.
 * See https://docs.openzeppelin.com/contracts/4.x/upgradeable#storage_gaps
 */
uint256[49] private __gap;
```

# Cases

# OpenZeppelin CVE-2021-41264

- OpenZeppelin 4.1.0 < 4.3.2 had a critical vuln that allowed to brick the proxy by directly initializing the implementation
- It existed in UUPS contract in the function `upgradeToAndCall` which could be called directly
- This function updates the implementation address in the proxy and atomically executes any migration/initialization function using DELEGATECALL
- But what if a target contract executes SELFDESTRUCT?

https://forum.openzeppelin.com/t/uupsupgradeable-vulnerability-post-mortem/15680

# OpenZeppelin CVE-2021-41264

- If this happens, the DELEGATECALL caller will be destroyed, i.e. the current active implementation contract
- Normally, we should not bother about it since onlyOwner can call `upgradeToAndCall`
- But if implementation contract is initialized directly this check is bypassed

```
modifier onlyProxy() {
    require(address(this) != __self, "Function must be called through delegatecall");
    require(_getImplementation() == __self, "Function must be called through active proxy");
    _;
}
```

# Wormhole

- Cross-chain bridge with >500M $ TVL
- Was hacked in early February, 325M $ lost (non-proxy issue)
- Another critical vuln similar to the OpenZeppelin's was submitted later in February by a whitehat via Immunefi
- Bug bounty – 10,000,000 $ 🤯

# Wormhole

- Vulnerability in Wormhole was possible due to the custom upgrade logic similar to the vulnerable OpenZeppelin < 4.3.2
- Wormhole used UUPS-style proxy
- A proxy upgrade was executed only if valid signatures of trusted addresses (called Guardians) were passed
- Since `upgradeTo` could be called directly and implementation was not initialized, it was possible to submit own set of Guardians and brick the proxy via SELFDESTRUCT in the new implementation

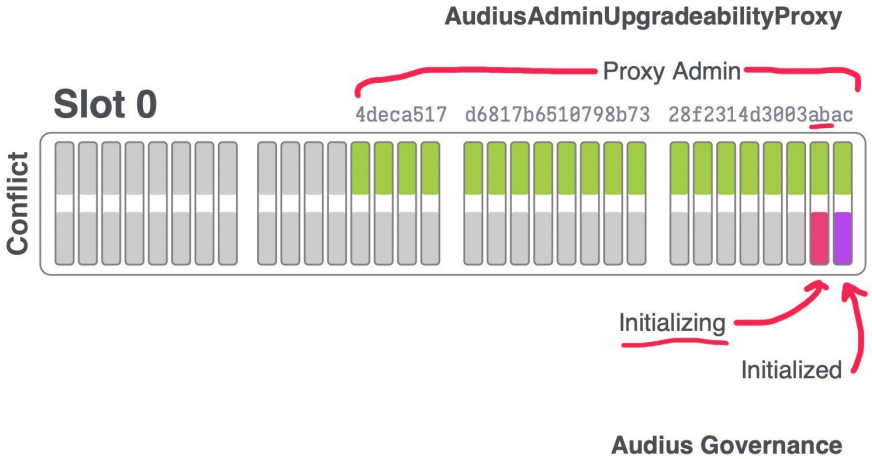
https://medium.com/immunefi/wormhole-uninitialized-proxy-bugfix-review-90250c41a43a

# Audius

- Audius - web3 Spotify
- Governance contract was behind a vulnerable custom proxy that inherited OpenZeppelin’s standard transparent proxy
- As a result Audius was hacked for 6,000,000 $
- Fun fact: contract was audited by OpenZeppelin

# Audius

- Custom proxy defined a state var `proxyAdmin` which occupied the first slot in the storage
- It overlapped variables `initializing` and `initialized` of OpenZeppelin's `Initializable` contract




Credit: @danielvf

# Audius

```
/**
 * @dev Indicates that the contract is in the process of being initialized.
 */
bool private initializing;

/**
 * @dev Modifier to use in the initializer function of a contract.
 */
modifier initializer() {
  require(msg.sender == proxyAdmin, "Only proxy admin can initialize");
  require(initializing || isConstructor() || !initialized, "Contract instanc

  bool isTopLevelCall = !initializing;
  if (isTopLevelCall) {
    initializing = true;
    initialized = true;
  }

  _;

  if (isTopLevelCall) {
    initializing = false;
```

Credit: @danielvf

# Tools & techniques

# sol2uml

| <<Contract>><br>FiatTokenV2_1<br>0xa2327a938febf5fec13bacfb16ae10ecbc4cbdcf | slot | type: <inherited contract>.variable (bytes) | |
|---|---|---|---|
| | 0 | address: Ownable._owner (20) | |
| | 1 | address: Pausable.pauser (20) | bool: Pausable.paused (1) |
| | 2 | address: Blacklistable.blacklister (20) | |
| | 3 | mapping(address=>bool): Blacklistable.blacklisted (32) | |
| | 4 | string: FiatTokenV1.name (32) | |
| | 5 | string: FiatTokenV1.symbol (32) | |
| | 6 | uint8: FiatTokenV1.decimals (1) | |
| | 7 | string: FiatTokenV1.currency (32) | |
| | 8 | address: FiatTokenV1.masterMinter (20) | bool: FiatTokenV1.initialized (1) |
| | 9 | mapping(address=>uint256): FiatTokenV1.balances (32) | |
| | 10 | mapping(address=>mapping(address=>uint256)): FiatTokenV1.allowed (32) | |
| | 11 | uint256: FiatTokenV1.totalSupply_ (32) | |
| | 12 | mapping(address=>bool): FiatTokenV1.minters (32) | |
| | 13 | mapping(address=>uint256): FiatTokenV1.minterAllowed (32) | |
| | 14 | address: Rescuable._rescuer (20) | |
| | 15 | bytes32: EIP712Domain.DOMAIN_SEPARATOR (32) | |
| | 16 | mapping(address=>mapping(bytes32=>bool)): EIP3009._authorizationStates (32) | |
| | 17 | mapping(address=>uint256): EIP2612._permitNonces (32) | |
| | 18 | uint8: FiatTokenV2._initializedVersion (1) | |

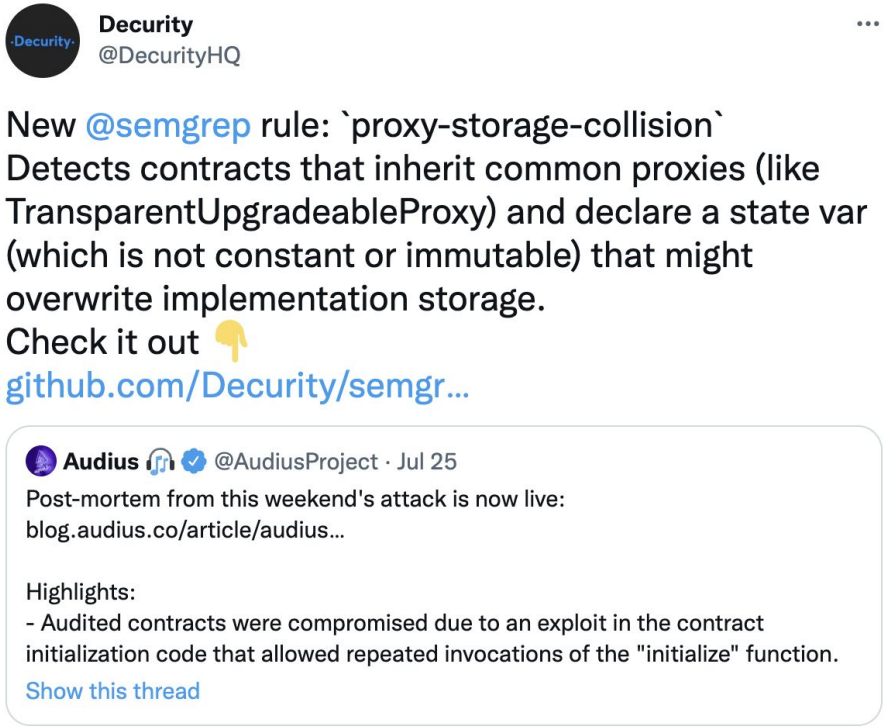
https://github.com/naddison36/sol2uml

# slither-check-upgradeability


https://github.com/crytic/slither

Credit: @ashekhirin

# proxy-storage-collision




https://github.com/Decurity/semgrep-smart-contracts



# Thank you!

Arseniy Reutov

**CTO Decurity.io**