10/7 - 12PM PST / 7PM UTC

Workshop #5

Secure Development by OpenZeppelin

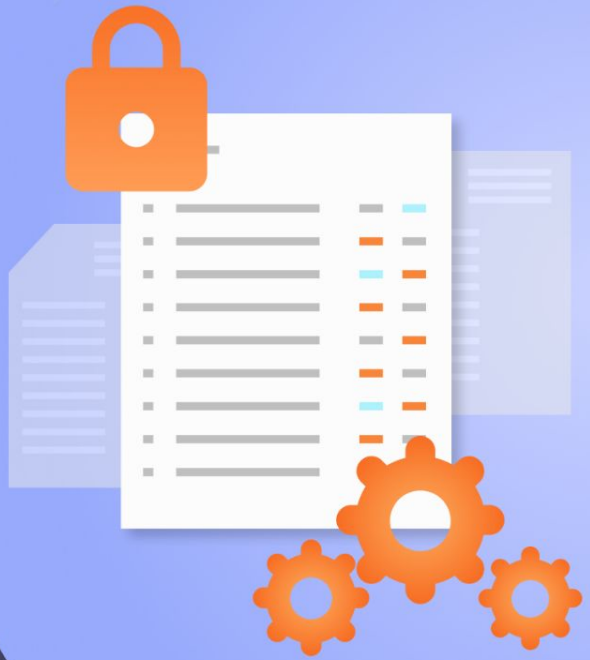
Security in Upgrades of Smart Contracts

Martin Abbatemarco

Security Researcher at OpenZeppelin

REGISTRATION REQUIRED

LIMITED TO 50 ATENDEES



Security solutions for industry leaders

Our mission is to protect the open economy

OpenZeppelin

Audits

200+ audits completed

Defender

3,000+ users in the first six months of launch, including many top DeFi projects

Contracts

$83B+ TVL in DeFi protocols, and thousands of NFTs including Beeple's **$69M** built on Contracts

ethereum foundation · celo · COSMOS · UNISWAP · Futureswap · MAKER · AAVE · Decentraland · brave · UMA · coinbase · Compound · Set · Balancer · augur · opyn · 1inch · InstaDApp · Optimism · GNOSIS · the graph · δY/δX · Polkadot.

Series of sessions

Secure Development

The dangers of token integration

Strategies for secure access controls

The dangers of price oracles

Strategies for secure governance

Security in upgrades of smart contracts

Onward with smart contract security

security in upgrades

what kind of upgrades ?

change contract parameters

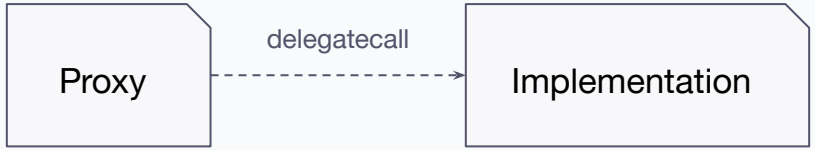
migrations

registries

strategies

→ **proxy-based upgrades** ←

proxy-based upgrades



eips.ethereum.org/EIPS/eip-897

https://blog.openzeppelin.com/proxy-patterns/

blog.gnosis.pm/solidity-delegateproxy-contracts-e09957d0f201

docs.openzeppelin.com/openzeppelin/upgrades

blog.openzeppelin.com/the-state-of-smart-contract-upgrades

and many, many, many, more ...

How hard can it be ?

well, it’s not easy

UUPSUpgradeable Vulnerability Post-mortem

General · Announcements

spalladino OpenZeppelin Team 96 2 12d

In early September, we received two independent reports of a vulnerability in the UUPSUpgradeable base contract of the OpenZeppelin Contracts library, first released 6 in version 4.0 in April, 2021.

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

Breaking Aave Upgradeability

POST DECEMBER 16, 2020 LEAVE A COMMENT

https://blog.trailofbits.com/2020/12/16/breaking-aave-upgradeability/

Malicious backdoors in Ethereum Proxies

A detailed explanation on how the Proxy pattern for smart contract upgradeability can be exploited.

Patricio Palladino Follow
Jun 1, 2018 · 5 min read

https://medium.com/nomic-labs-blog/malicious-backdoors-in-ethereum-proxies-62629adf3357

High severity

[H01] Corruptible storage upgradeability pattern

https://blog.openzeppelin.com/ribbon-finance-audit/

USDC v2: Upgrading a multi-billion dollar ERC-20 token

Coinbase Follow
Dec 31, 2020 · 10 min read

https://blog.coinbase.com/usdc-v2-upgrading-a-multi-billion-dollar-erc-20-token-b57cd9437096

Safety Module Outage

X Foundation

https://dydx.foundation/blog/en/outage-1

⚠ WARNING

Violating any of these storage layout restrictions will cause the upgraded version of the contract to have its storage values mixed up, and can lead to critical errors in your application.

https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable#modifying-your-contracts

Beware of the proxy: learn how to exploit function clashing

Security

tinchoabbate OpenZeppelin Team 1 Jul '19

https://forum.openzeppelin.com/t/beware-of-the-proxy-learn-how-to-exploit-function-clashing/1070

the unknown unknowns

Problem(s) ?

(2 minutes)

```
/**
 * @notice To be called in an emergency by the owner. In initial versions, multisig. Then governance.
           The function takes all tokens out, incentivizing with ETH according to how much tokens were saved.
           Any remaining ETH is sent to the owner.
           Finally, the upgrade is triggered.
 */
function emergencyUpgrade(address newImplementation, address recipient) public onlyOwner {
    uint256 tokenBalance = token.balanceOf(address(this));

    // Transfer out deposited tokens to the owner
    token.transfer(owner(), tokenBalance);

    // Calculate how much ETH the tokens are worth, and send it out
    uint256 amount = oracle.getPrice(token) * tokenBalance;
    payable(recipient).sendValue(amount);

    // Any remaining ETH goes to the owner
    if(address(this).balance > 0) {
        payable(owner()).sendValue(address(this).balance);
    }

    // Trigger the contract upgrade
    upgradeTo(newImplementation);
}
```

question the code

some initial triggers

```
/**
 * @notice To be called in an emergency by the owner. In initial versions, multisig. Then governance.
           The function takes all tokens out, incentivizing with ETH according to how much tokens were saved.
           Any remaining ETH is sent to the owner.
           Finally, the upgrade is triggered.
 */
function emergencyUpgrade(address newImplementation, address recipient) public onlyOwner {
    uint256 tokenBalance = token.balanceOf(address(this));

    // Transfer out deposited tokens to the owner
    token.transfer(owner(), tokenBalance);

    // Calculate how much ETH the tokens are worth, and send it out
    uint256 amount = oracle.getPrice(token) * tokenBalance;
    payable(recipient).sendValue(amount);

    // Any remaining ETH goes to the owner
    if(address(this).balance > 0) {
        payable(owner()).sendValue(address(this).balance);
    }

    // Trigger the contract upgrade
    upgradeTo(newImplementation);
}
```

What's behind onlyOwner now ? Trust assumptions ?

What kind of tokens ? Could transfer fail ?

Can the owner handle tokens ? And ETH ?

What kind of oracle ?

Allowed to read ? Is token supported ? Units ? Price zero ?

What if no ETH is deposited ?

Is recipient of ETH trusted ? DoS ? Reentrancy ?

And about that **upgradeTo** function call ...

some initial triggers

```
// Trigger the contract upgrade
upgradeTo(newImplementation);
```

What upgradeability pattern is being used ?

Does new implementation respect storage layouts ?

Is new implementation correctly initialized ?

Does new implementation need upgrade logic ?

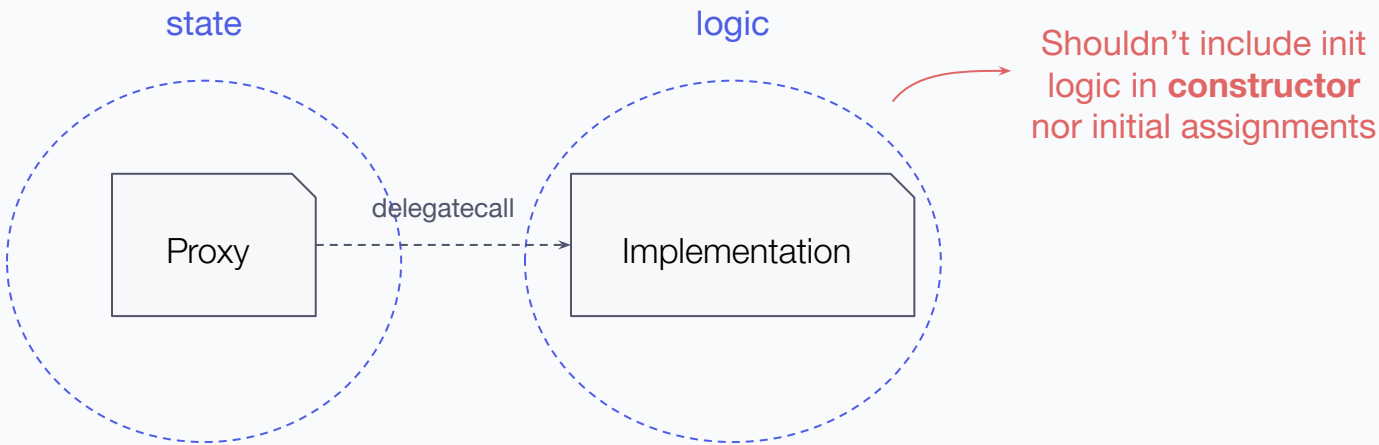
Function clashing between implementation and proxy ?

Are there any safety checks to avoid bricking the upgrade ?

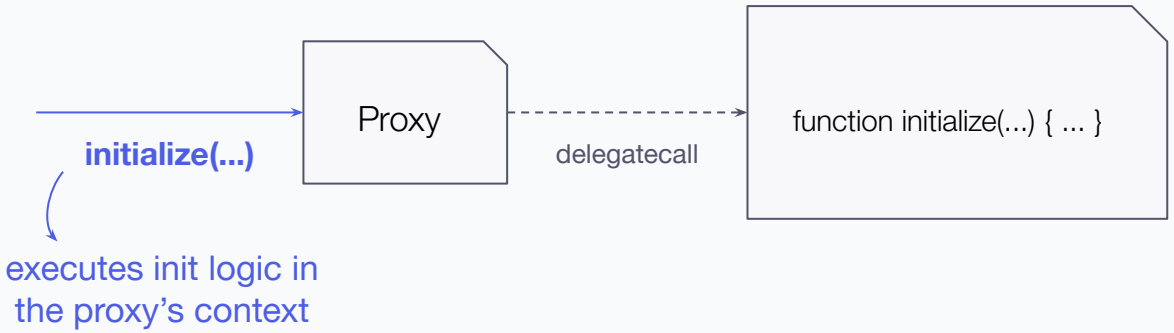
What are the off-chain validations on the whole process ?

initializers

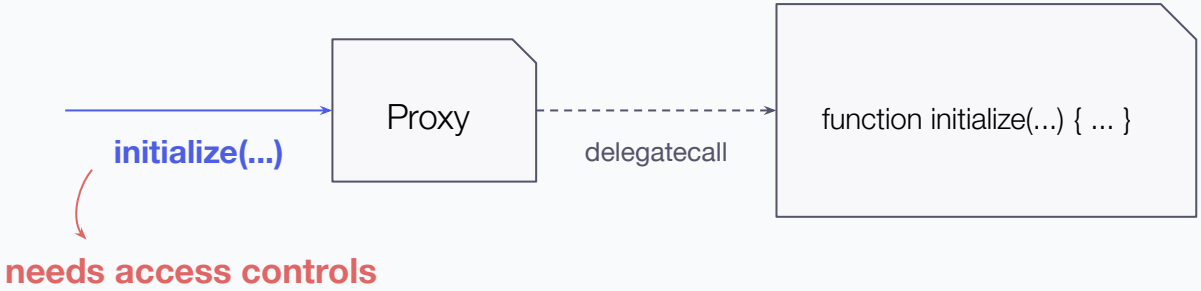
```
contract Implementation {
    uint256 public constant a = 42; // OK
    uint256 public b = 1; // NOT OK
    address public owner;

    constructor(address newOwner) { // NOT OK
        owner = newOwner;
    }
}
```



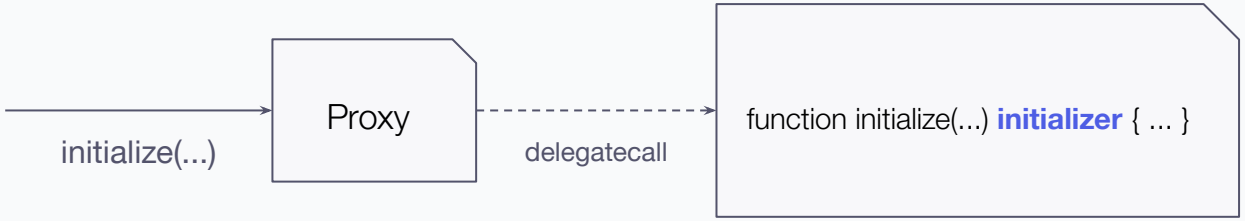
```
contract Implementation {
    uint256 public constant a = 42; // OK
    uint256 public b;
    address public owner;

    function initialize(address newOwner) external { // BETTER (BUT STILL NOT OK)
        b = 1;
        owner = newOwner;
    }
}
```



```
contract Implementation {
    uint256 public constant a = 42; // OK
    uint256 public b;
    address public owner;

    function initialize(address newOwner) external { // BETTER (BUT STILL NOT OK)
        b = 1;
        owner = newOwner;
    }
}
```



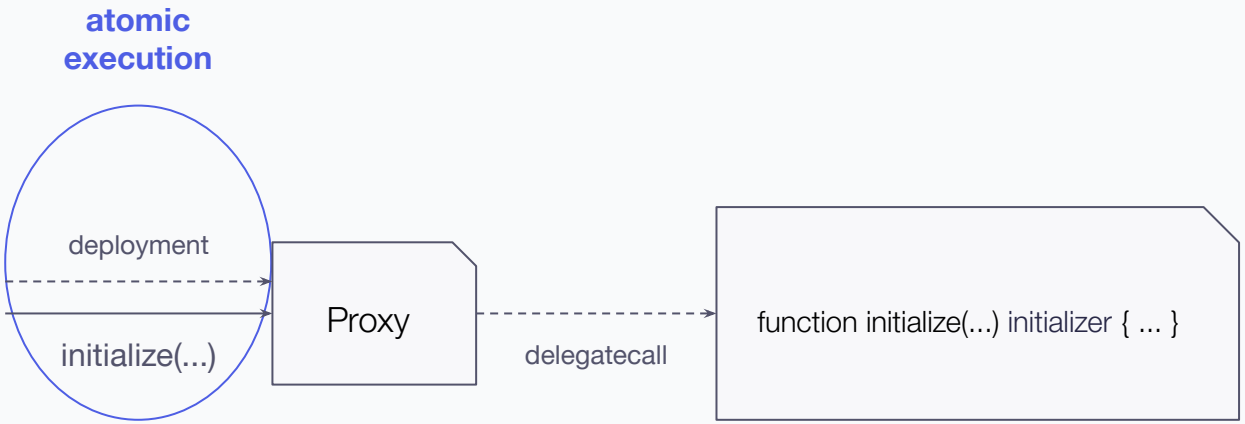
```
import "@openzeppelin/contracts/proxy/utils/Initializable.sol";

contract Implementation is Initializable {
    uint256 public constant a = 42; // OK
    uint256 public b;
    address public owner;

    function initialize(address newOwner) external initializer {
        b = 1;
        owner = newOwner;
    }
}
```



ensure initialize is only called once

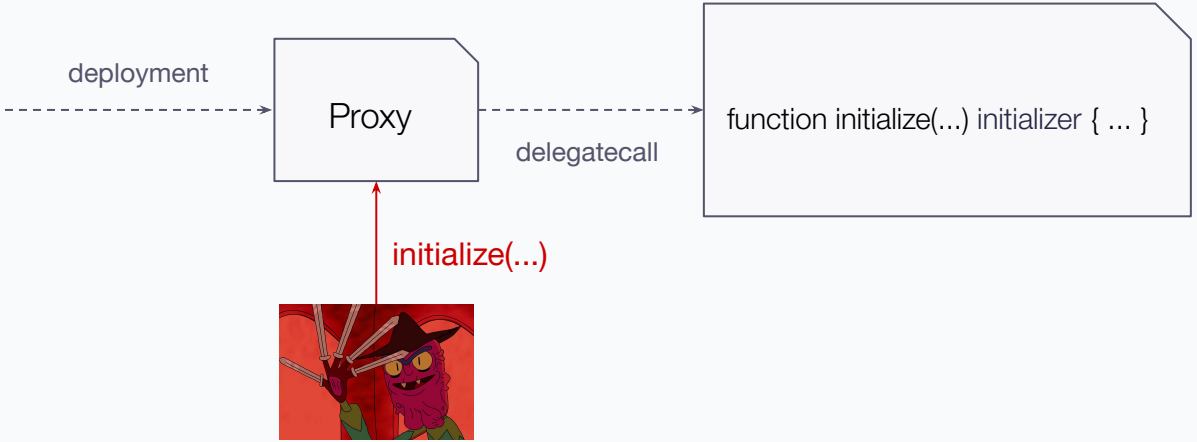
but who ? when ?

See Initializable contract at https://docs.openzeppelin.com/contracts/4.x/api/proxy#Initializable

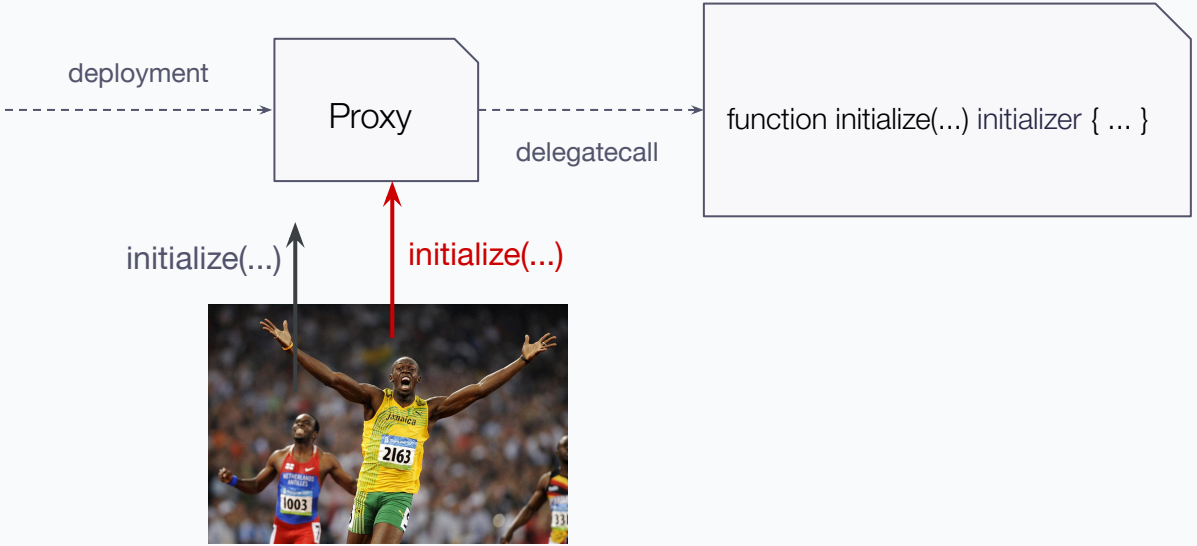

atomic
execution
deployment
initialize(...)
Proxy
delegatecall
function initialize(...) initializer { ... }

what can wrong with initializers ?

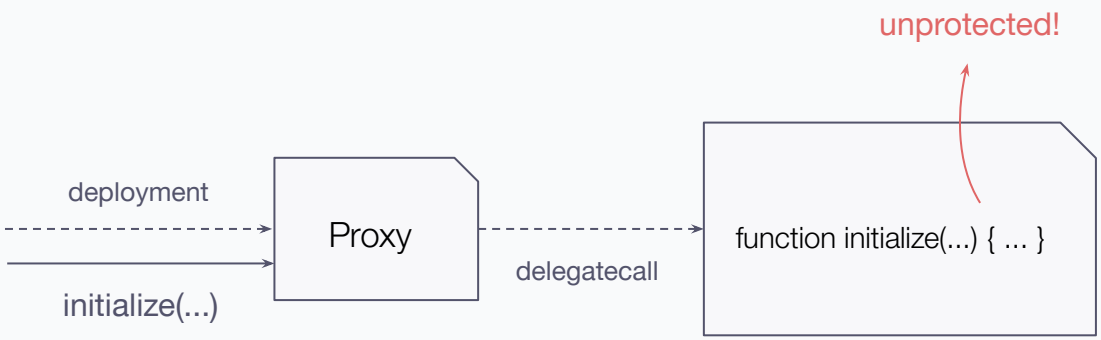
uninitialized proxies



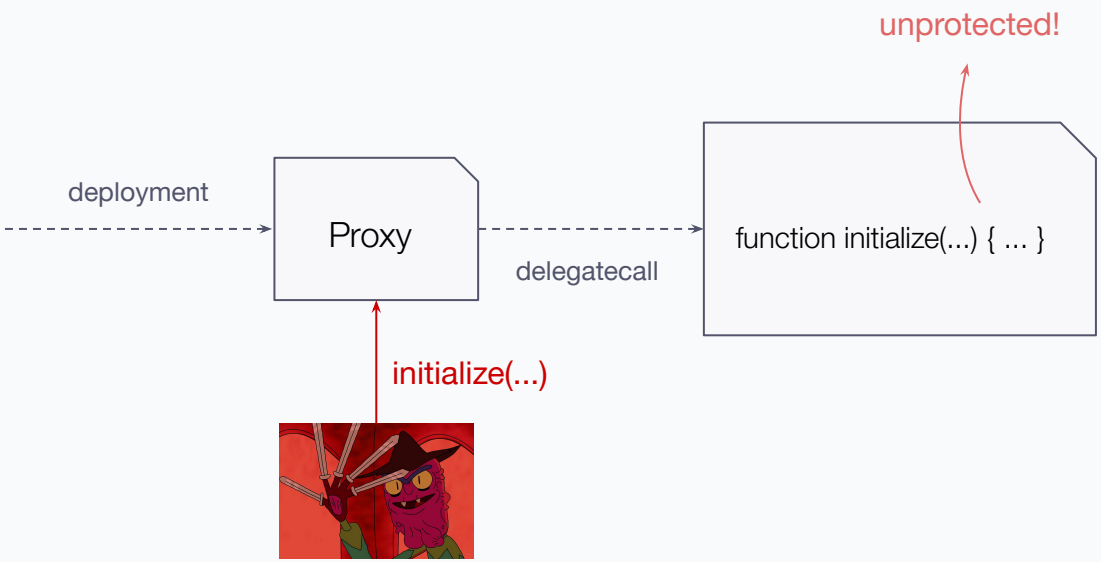
uninitialized proxies



reinitialized proxies

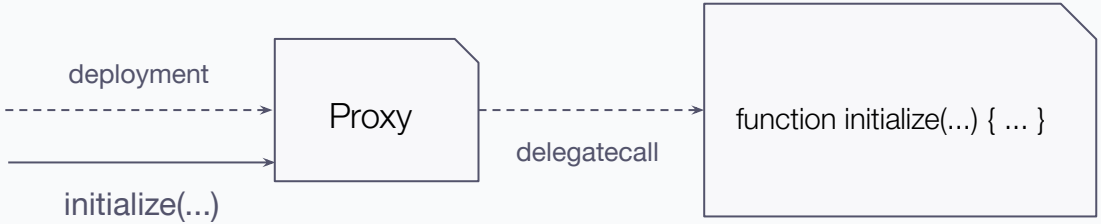


reinitialized proxies



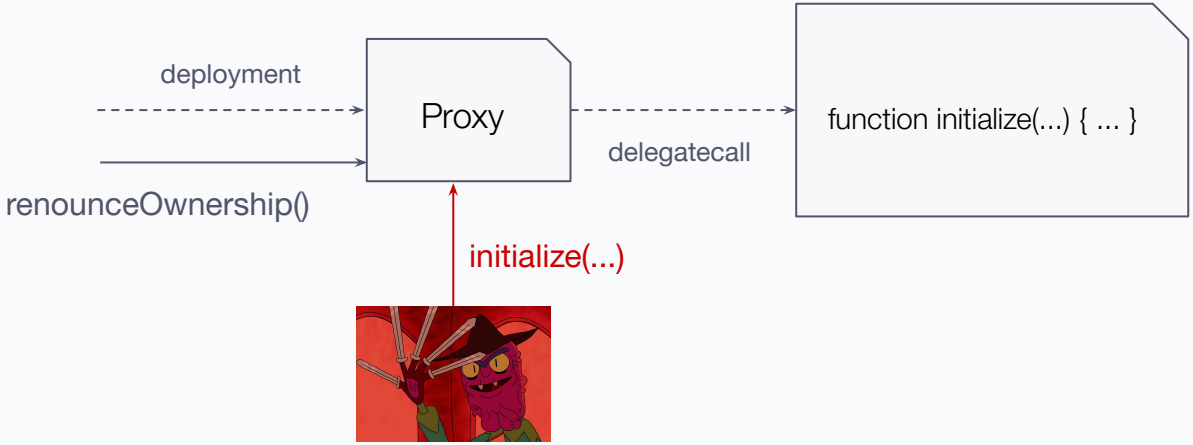
```
function initialize(address newOwner) external {
    require(newOwner != address(0));
    require(owner == address(0));

    owner = newOwner;
}

function renounceOwnership() external {
    require(msg.sender == owner);
    owner = address(0);
}
```

reinitialized proxies



```
function initialize(address newOwner) external {
    require(newOwner != address(0));
    require(owner == address(0));

    owner = newOwner;
}

function renounceOwnership() external {
    require(msg.sender == owner);
    owner = address(0);
}
```

reinitialized proxies



incompatible inheritance

```
contract Ownable {
    address public owner;
    constructor() {
        owner = msg.sender;
    }
}

contract Implementation is Initializable, Ownable {
    uint256 public number;
    function initialize(uint256 n) external initializer {
        number = n;
    }
}
```

NOT OK

```
import "@openzeppelin/contracts/proxy/utils/Initializable.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

contract Implementation is Initializable, Ownable {
    uint256 public number;
    function initialize(uint256 n) external initializer {
        number = n;
    }
}
```

incompatible inheritance

BETTER!
BUT NOT OK

```
import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";

contract Implementation is OwnableUpgradeable {
    uint256 public number;
    function initialize(uint256 n) external initializer {
        number = n;
    }
}
```

https://docs.openzeppelin.com/contracts/4.x/upgradeable

initialization chains

```
import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";

contract Implementation is OwnableUpgradeable {
    uint256 public number;
    function initialize(uint256 n) external initializer {
        number = n;
    }
}
```

What's missing ?

initialization chains

```
import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";

contract Implementation is OwnableUpgradeable {
    uint256 public number;
    function initialize(uint256 n) external initializer {
        __Ownable_init();
        number = n;
    }
}
```

initializes parent
(setting msg.sender as owner)

initialization chains

id = 1

```
abstract contract Base  {
    uint256 public id;

    constructor() {
        id++;
    }
}

abstract contract A is Base { }

abstract contract B is Base { }

contract Implementation is A, B { }
```

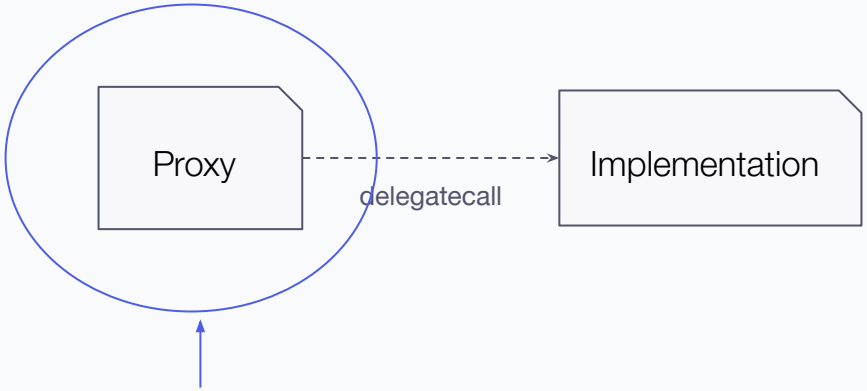
id = 2

```
import "@openzeppelin/contracts/proxy/utils/Initializable.sol";

abstract contract Base is Initializable {
    uint256 public id;

    function __Base_init() internal initializer {
        id++;
    }
}

abstract contract A is Base {
    function __A_init() internal initializer {
        __Base_init();
    }
}

abstract contract B is Base {
    function __B_init() internal initializer {
        __Base_init();
    }
}

contract Implementation is A, B {
    function initialize() external initializer {
        __A_init();
        __B_init();
    }
}
```

https://docs.openzeppelin.com/contracts/4.x/upgradeable#multiple-inheritance

bricked upgrades

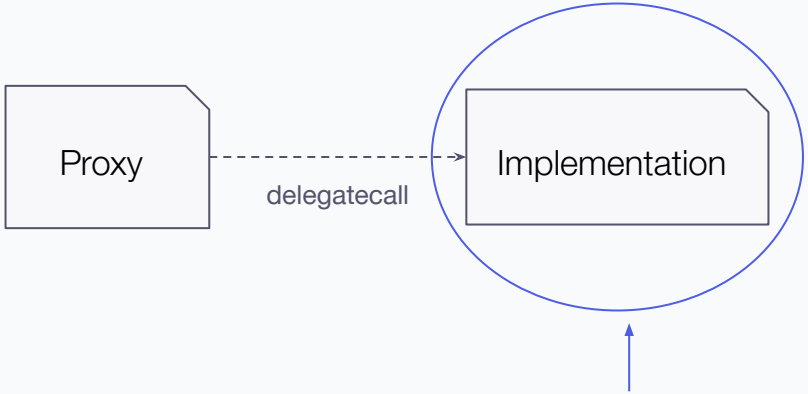
Transparent Proxy Pattern



Where do we put the **logic for upgrading** ?

blog.openzeppelin.com/the-state-of-smart-contract-upgrades/#transparent-proxies

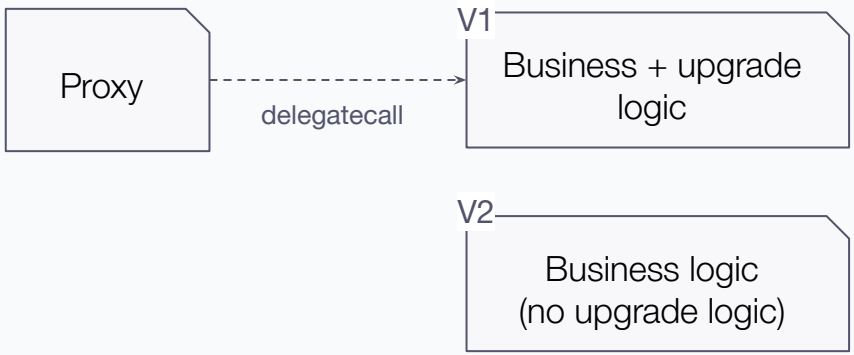
UUPS Pattern



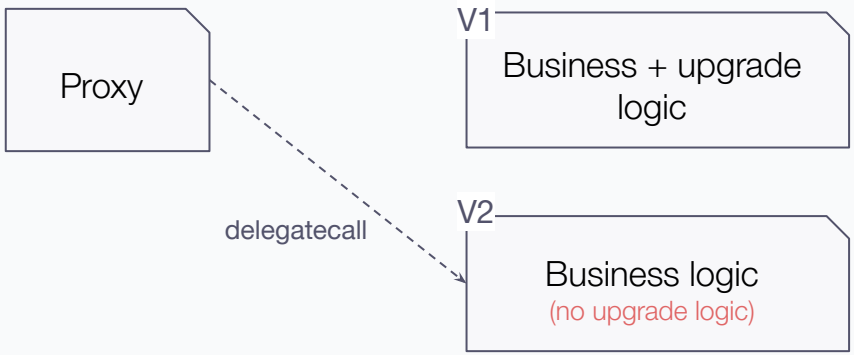
Where do we put the **logic for upgrading** ?

blog.openzeppelin.com/the-state-of-smart-contract-upgrades/#universal-upgradeable-proxies

bricked upgrades



bricked upgrades



Can’t upgrade anymore

bricked upgrades

(1) Do upgrade and setup

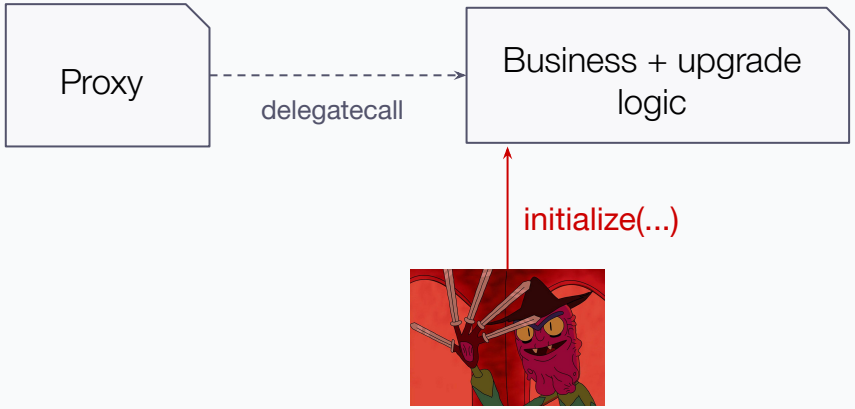
(2) Do rollback

(3) Confirm upgrade

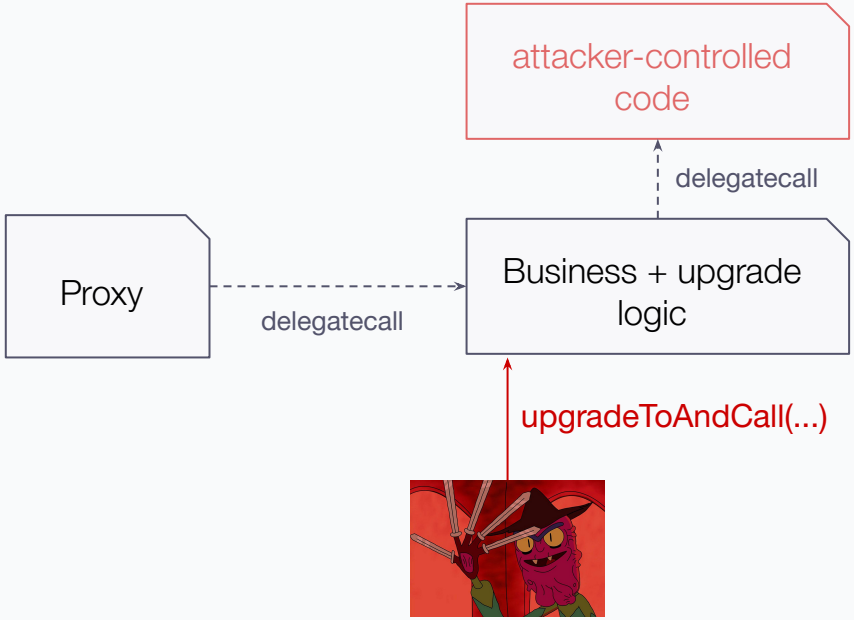
```
function _upgradeToAndCallSecure(
    address newImplementation,
    bytes memory data,
    bool forceCall
) internal {
    address oldImplementation = _getImplementation();

    // Initial upgrade and setup call
    _setImplementation(newImplementation);
    if (data.length > 0 || forceCall) {
        Address.functionDelegateCall(newImplementation, data);
    }

    // Perform rollback test if not already in progress
    StorageSlot.BooleanSlot storage rollbackTesting = StorageSlot.getBooleanSlot(_ROLLBACK_SLOT);
    if (!rollbackTesting.value) {
        // Trigger rollback using upgradeTo from the new implementation
        rollbackTesting.value = true;
        Address.functionDelegateCall(
            newImplementation,
            abi.encodeWithSignature("upgradeTo(address)", oldImplementation)
        );
        rollbackTesting.value = false;
        // Check rollback was effective
        require(oldImplementation == _getImplementation(), "ERC1967Upgrade: upgrade breaks further upgrades");
        // Finally reset to the new implementation and log the upgrade
        _upgradeTo(newImplementation);
    }
}
```

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.3.2/contracts/proxy/ERC1967/ERC1967Upgrade.sol

uninitialized implementations

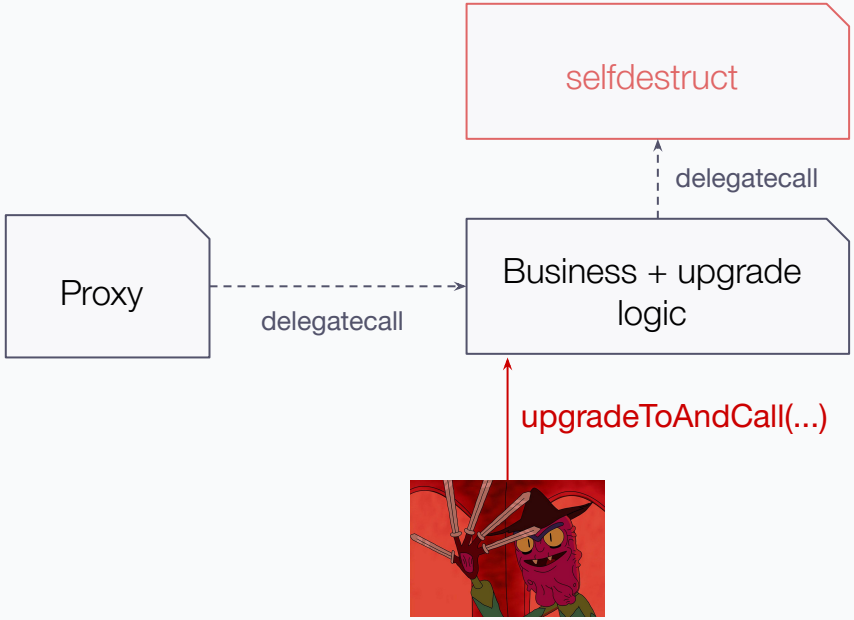


uninitialized implementations



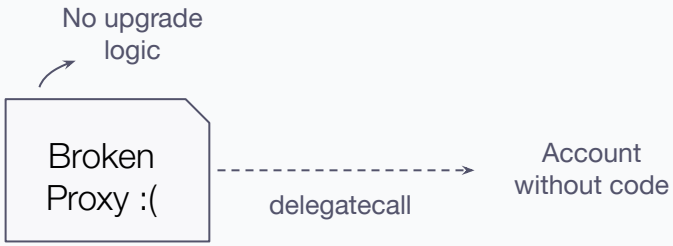
https://forum.openzeppelin.com/t/uupsupgradeable-vulnerability-post-mortem/15680

uninitialized implementations



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

uninitialized implementations



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

storage layout

solc --storage-layout Proxy.sol

```
"storage": [
    {
        "label": "implementation",
        "slot": "0",
        "type": "t_address"
    },
    {
        "label": "owner",
        "slot": "1",
        "type": "t_address"
    }
]
```

solc --storage-layout Implementation.sol

```
"storage": [
    {
        "label": "initialized",
        "slot": "0",
        "type": "t_bool"
    },
    {
        "label": "number",
        "slot": "1",
        "type": "t_uint256"
    }
]
```

```
address public implementation;
address public owner;
```

```
bool public initialized;
uint256 public number;
```

r/w `implementation` slot ← r/w `initialized`

r/w `owner` slot ← r/w `number`

STORAGE COLLISION 😱

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

Avoiding proxy-implementation storage collisions

```
|Proxy                      |Implementation           |
|---------------------------|-------------------------|
|...                        |bool initialized         |
|...                        |uint256 number           |
|...                        |...                      |
|...                        |...                      |
|...                        |                         |
|...                        |                         |
|...                        |                         |
|...                        |                         |
|address implementation     |                         | <=== Randomized slot
|...                        |                         |
|...                        |                         |
|address owner              |                         | <=== Randomized slot
```

eips.ethereum.org/EIPS/**eip-1967**

docs.openzeppelin.com/upgrades-plugins/1.x/proxies#unstructured-storage-proxies

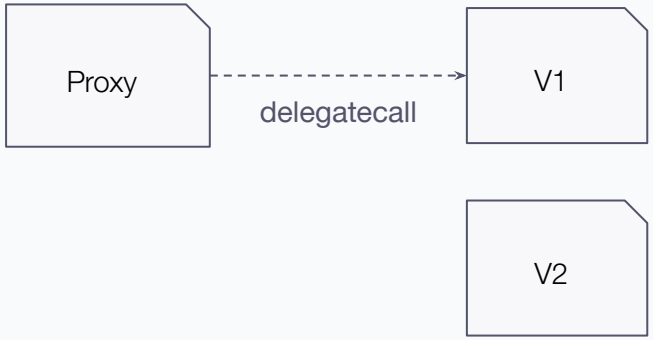
blog.openzeppelin.com/the-state-of-smart-contract-upgrades/#unstructured-storage

Avoiding proxy-implementation storage collisions

```
assert(ADMIN_SLOT == bytes32(uint256(keccak256("eip1967.proxy.admin")) - 1));
assert(
    IMPLEMENTATION_SLOT == bytes32(uint256(keccak256("eip1967.proxy.implementation")) - 1)
);
```

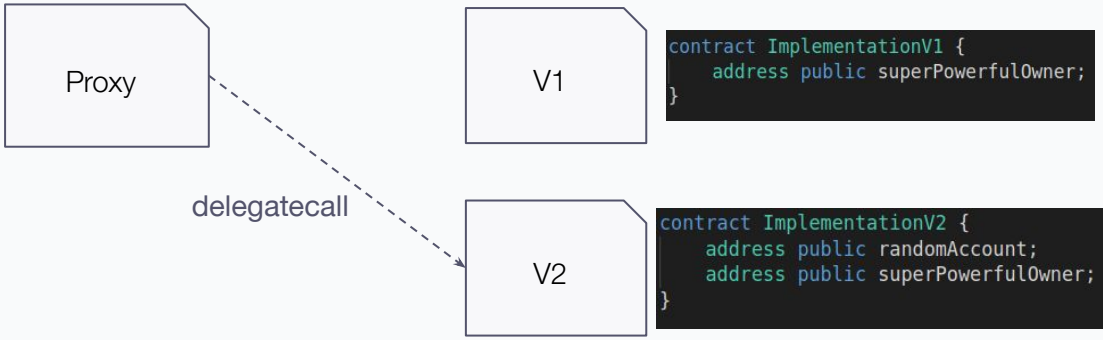
https://github.com/graphprotocol/contracts/blob/v1.8.0/contracts/upgrades/GraphProxy.sol#L41

storage collisions between versions




```
contract ImplementationV1 {
    address public superPowerfulOwner;
}
```

storage collisions between versions



randomAccount is at slot 0 now !

You MUST preserve storage layouts across upgrades

You MUST preserve storage layouts across upgrades

- Cannot change type
- Cannot change order declaration
- Cannot introduce new vars before existing ones
- Cannot remove existing ones
- Always append new vars at the end
  - But not in base contracts! :)

https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable#modifying-your-contracts

That's why you'll see things like

```
uint256[45] private __gap;
```

https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/v4.3.2/contracts/token/ERC20/ERC20Upgradeable.sol

```
// Reserved storage space to allow for layout changes in the future.
uint256[50] private ______gap;
```

https://github.com/aave/protocol-v2/blob/master/contracts/protocol/libraries/aave-upgradeability/VersionedInitializable.sol#L76

```
contract ComptrollerV2Storage is ComptrollerV1Storage {
contract ComptrollerV3Storage is ComptrollerV2Storage {
contract ComptrollerV4Storage is ComptrollerV3Storage {
contract ComptrollerV5Storage is ComptrollerV4Storage {
contract Comptroller is ComptrollerV5Storage,
```

https://github.com/compound-finance/compound-protocol/blob/master/contracts/ComptrollerStorage.sol

function clashing

does this compile ?

```
pragma solidity ^0.8.0;

contract Example {
    function AcoraidaMonicaWantsToKeepALogOfTheWinner(address) external {}
    function upgrade(address) external {}
}
```

```
Error: Function signature hash collision for upgrade(address)
 --> <stdin>:1:25:
  |
1 | pragma solidity ^0.8.0; contract Example { function AcoraidaMonicaWantsToKee
pALogOfTheWinner(address) external {} function upgrade(address) external {} }
  |                         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

Search Signatures 0x0900f010 Search

ID	Text Signature	Bytes Signature
150132	AcoraIdaMonicaWantsToKeepALogOfTheWinner(address)	0x0900f010
6954	upgrade(address)	0x0900f010

https://www.4byte.directory/signatures/?bytes4_signature=0x0900f010

Search Signatures 0xa9059cbb Search

ID	Text Signature	Bytes Signature
161159	transfer(bytes4[9],bytes5[6],int48[11])	0xa9059cbb
31780	many_msg_babbage(bytes1)	0xa9059cbb
145	transfer(address,uint256)	0xa9059cbb

https://www.4byte.directory/signatures/?bytes4_signature=0xa9059cbb

Good luck trying to upgrade this

```
contract Proxy {
    function AcoraidaMonicaWantsToKeepALogOfTheWinner(address) external {
        // do something
    }

    fallback() {
        // delegate to implementation
    }
}
```

```
contract Implementation {
    function upgrade(address) external {
        // do upgrade
    }
}
```

Beware of the proxy: learn how to exploit function clashing

Security

tinchoabbate OpenZeppelin Team 1 Jul '19

Could be used for evil, beware!

forum.openzeppelin.com/t/beware-of-the-proxy-learn-how-to-exploit-function-clashing/1070

access controls

Who upgrades ?

Who upgrades ?

Smart Contract Security Guidelines

Smart Contract Security Guidelines #1: The Dangers of Token Integration

The more DeFi layers we integrate and compose, the more careful we need to be. As the stack grows, so does systemic risk.

Smart Contract Security Guidelines #2: Strategies for Secure Access Controls

Learn about strategies to consider when designing access controls in your system

Smart Contract Security Guidelines #3: The Dangers of Price Oracles

This guide focuses on showcasing the architecture, roles and subtleties of most popular price oracles in Ethereum, ways to safely integrate them with defensive programming practices.

https://blog.openzeppelin.com/smart-contract-security-guidelines/

tooling

OpenZeppelin | docs

Upgrades Plugins

Integrate upgrades into your existing workflow. Plugins for Hardhat and Truffle to deploy and manage upgradeable contracts on Ethereum.

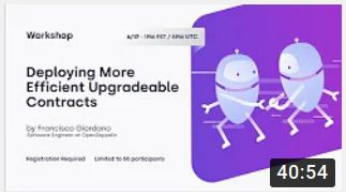
docs.openzeppelin.com/upgrades-plugins

OpenZeppelin | docs

Upgrading a contract via a multisig

docs.openzeppelin.com/defender/guide-upgrades

Managing Smart Contract Upgrades with Defender



Deploying More Efficient Upgradeable Contracts

www.youtube.com/OpenZeppelin

SLITHER

Upgradeability Checks

`slither-check-upgradeability` helps review contracts that use the delegatecall proxy pattern.

github.com/crytic/slither/wiki/Upgradeability-Checks

github.com/crytic/slither/wiki/Detector-Documentation#unprotected-upgradeable-contract

On upgrades

Closing thoughts

1. You'll likely need to upgrade your project. Choose how.
2. Proxy-based upgrades are here to stay. Be careful, not afraid.
3. The unknown unknowns are dangerous. Learn and share.
4. Several must-follow rules to get proxies right. Use available tooling.
5. Review and test contracts and their *transitions*

On upgrades

Where do I learn more ?

- blog.openzeppelin.com/proxy-patterns (old but relevant)
- “Writing upgradeable contracts” (zpl.in/upgrades-plugins)
- blog.openzeppelin.com/the-state-of-smart-contract-upgrades

Series of sessions

Secure Development

The dangers of token integration ✓

Strategies for secure access controls ✓

The dangers of price oracles ✓

Strategies for secure governance ✓

Secure smart contract upgrades ✓

Onward with smart contract security

We're hiring!

Open Roles

- Blockchain Security Engineer
- Full Stack Ethereum Developer
- Open Source Developer
- Site Reliability Engineer
- and more!

Check out more

zpl.in/join

Thanks!

Learn more

openzeppelin.com
defender.openzeppelin.com
blog.openzeppelin.com
forum.openzeppelin.com

Contact

@tinchoabbate
tincho@openzeppelin.com