OpenZeppelin

Managing Smart Contract Upgrades

with Defender and Upgrades Plugins

zpl.in/upgrades-workshop

Martin Verzilli

mverzilli@openzeppelin.com

@mverzilli

Workshop agenda

- Intro
  - OpenZeppelin and Defender
  - Background requirements
- Hands on
  - Create and deploy an upgradeable contract from scratch
  - Upgrade from your development environment
  - Transfer ownership to a Multisig
  - Upgrade from Defender Admin
- Q&A

OpenZeppelin

Our mission is to protect the open economy

OpenZeppelin is a software company that provides **security audits** and **products** for decentralized systems.

Projects from any size — from new startups to established organizations — trust OpenZeppelin to build, inspect and connect to the open economy.

Compound coinbase AAVE

δY/δX brave UMA

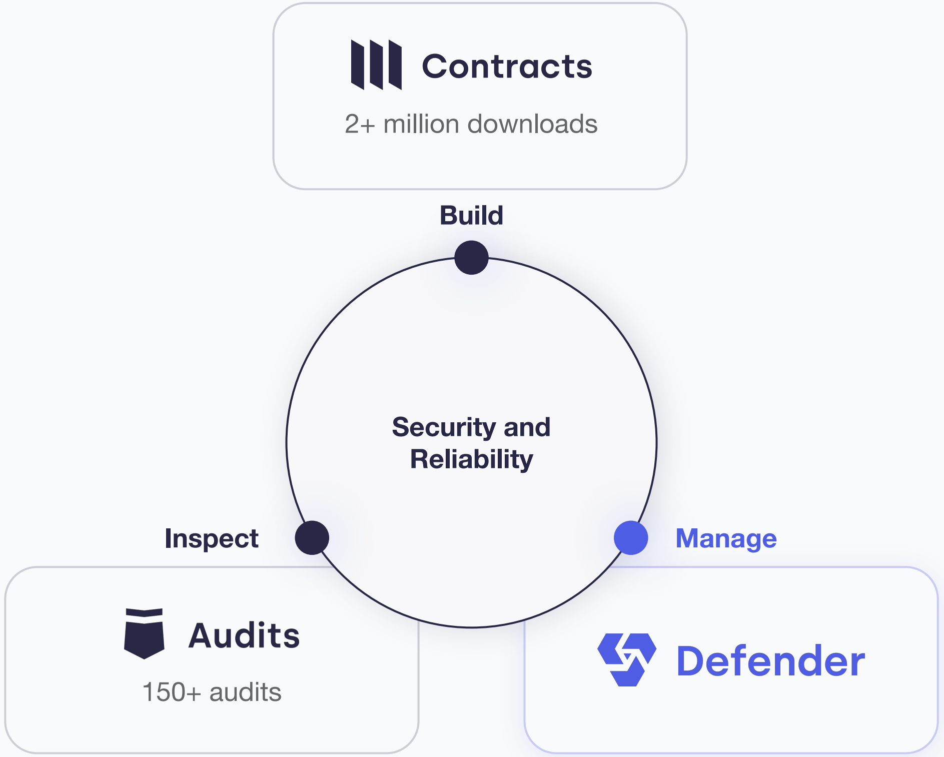
Balancer ethereum foundation augur

celo CIRCLE BitGo

pool together Set opyn

Security, Reliability and Risk Management

OpenZeppelin provides a complete suite of **security and reliability products** to build, manage, and inspect all aspects of software development and operations for Ethereum projects.



Defender features

- **Admin** - interface for contract administration
- **Relayer** - secure hosted keys and reliable transaction delivery
- **Autotasks** - serverless code for automated and off-chain logic
- **Sentinels** - monitor transactions, trigger notifications and Autotasks
- **Advisor** - best-practices in securing your decentralized system

Learn more: **https://docs.openzeppelin.com/defender/**

Technical background requirements for this workshop

- Basic programming skills.
- Basic understanding of smart contracts on Ethereum.
- Some familiarity with the Ethereum development ecosystem helps (Hardhat, Truffle, Etherscan), but is not required.

Why Upgrades? Because we’re not perfect!

- Someone’s eventually going to **find a vulnerability**.
- We’ll eventually want to **introduce enhancements**.

We’ll want to implement the changes with **minimal service disruption**.

Properties of upgradeable contracts

- We can replace their **implementation...**
- ...while preserving their **address** and **state**

But...

*"...now **admins** can pull the rug from under my feet. As a user, an upgradeable contract means I have to increase my level of trust in **external actors**."*

But...

> *"...now **admins** can pull the rug from under my feet. As a user, an upgradeable contract means I have to increase my level of trust in **external actors**."*

- ...**Absolutely true** (*in a vacuum*)!

But...

*"...now **admins** can pull the rug from under my feet. As a user, an upgradeable contract means I have to increase my level of trust in **external actors**."*

- ...**Absolutely true** (*in a vacuum*)!
- In practice, if you find about a vulnerability in your contract, where would you rather be? Where would your users rather be?

But...

*"...now **admins** can pull the rug from under my feet. As a user, an upgradeable contract means I have to increase my level of trust in **external actors**."*

- **...Absolutely true** (*in a vacuum*)!
- In practice, if you find about a vulnerability in your contract, where would you rather be? Where would your users rather be?
  - **Upgradeable world**: pause, code upgrade, deploy upgrade, unpause 😎

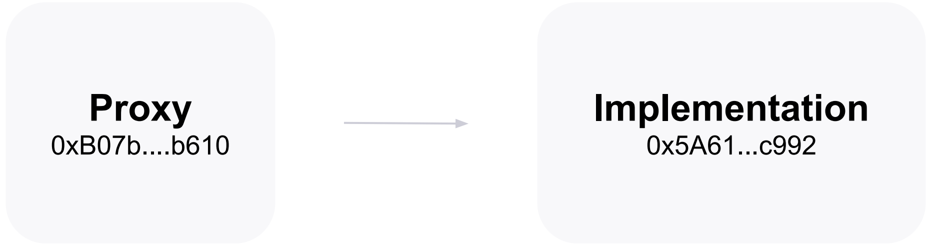
But…

*"…now **admins** can pull the rug from under my feet. As a user, an upgradeable contract means I have to increase my level of trust in **external actors**."*

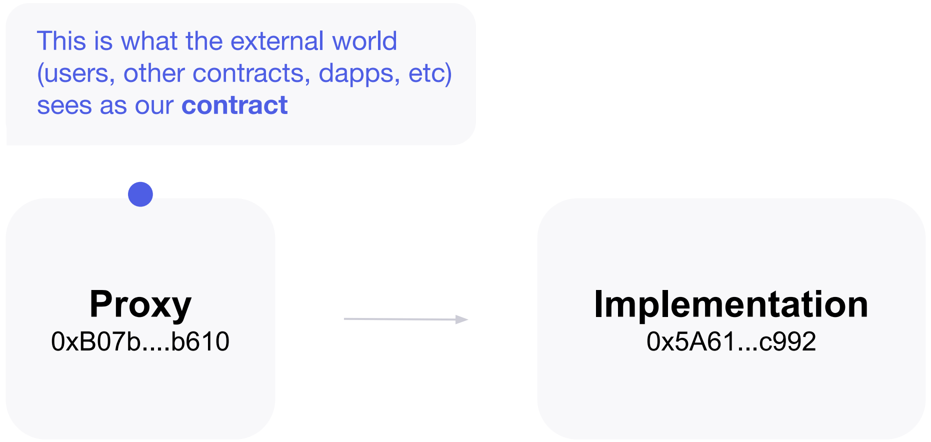
- …**Absolutely true** (*in a vacuum*)!
- In practice, if you find about a vulnerability in your contract, where would you rather be? Where would your users rather be?
  - **Upgradeable world**: pause, code upgrade, deploy upgrade, unpause 😎
  - **Non-upgradeable world:** pause, code new contract, deploy new contract, announce new address, migrate balances, ask users, dapps, other contracts to move to the new address, keep both running for a transition period, … 🥵

Extremely low resolution anatomy of an upgradeable contract

“All problems in computer science can be solved by another level of indirection”
Butler Lampson

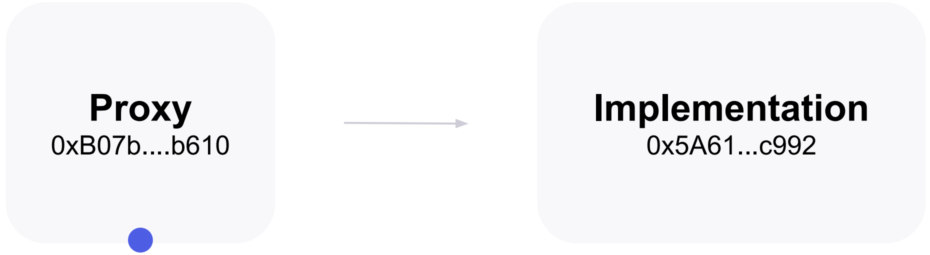


Extremely low resolution anatomy of an upgradeable contract


This is what the external world (users, other contracts, dapps, etc) sees as our **contract**
Proxy
0xB07b....b610
Implementation
0x5A61...c992

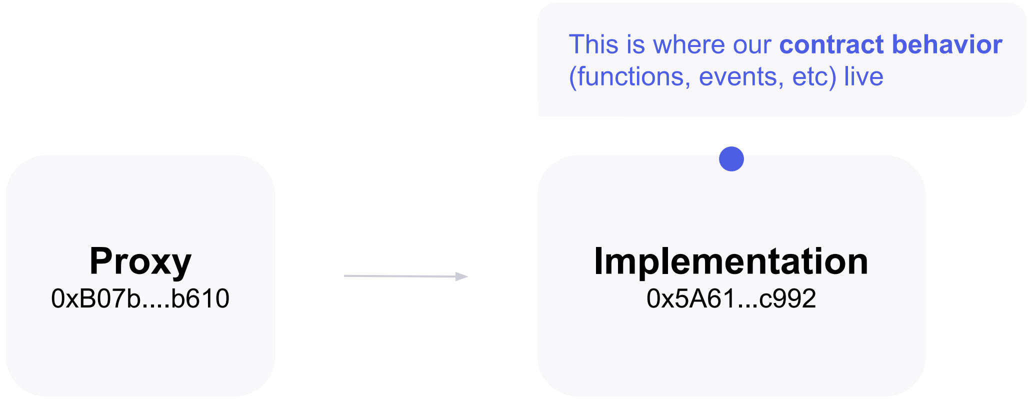
Extremely low resolution anatomy of an upgradeable contract

This is what the external world (users, other contracts, dapps, etc) sees as our **contract**

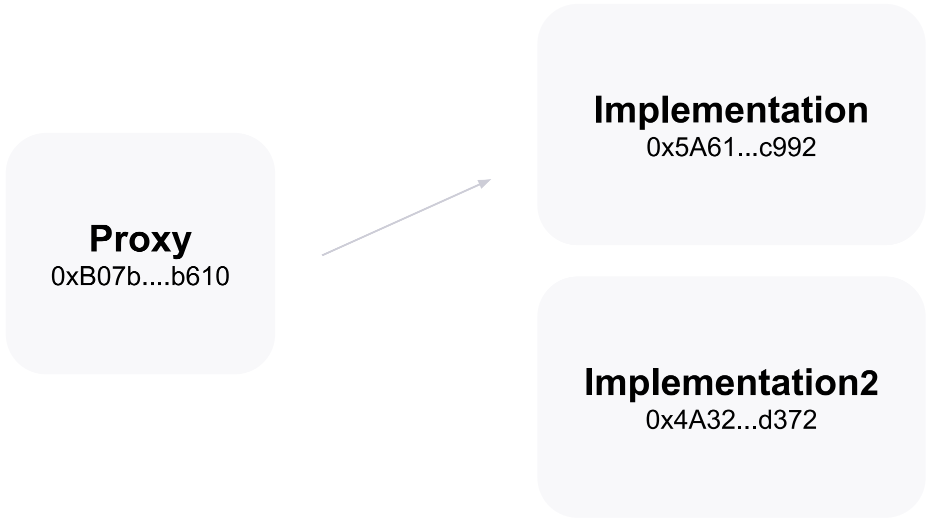


This is also where **balances and internal state** stays

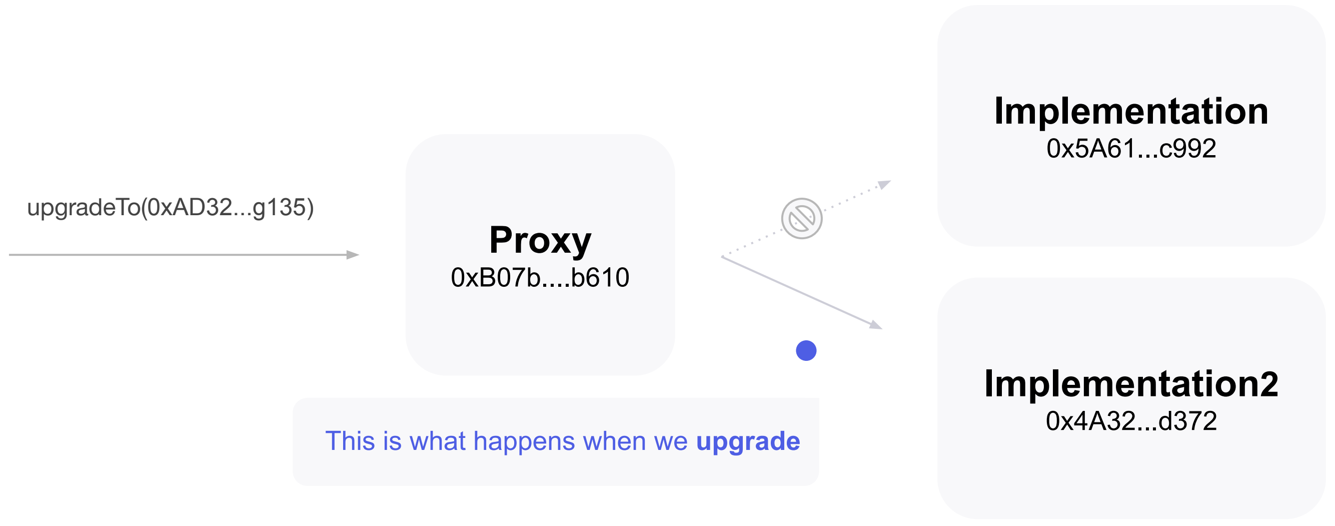
Extremely low resolution anatomy of an upgradeable contract


This is where our **contract behavior** (functions, events, etc) live
Proxy
0xB07b....b610
Implementation
0x5A61...c992

Extremely low resolution anatomy of an upgradeable contract

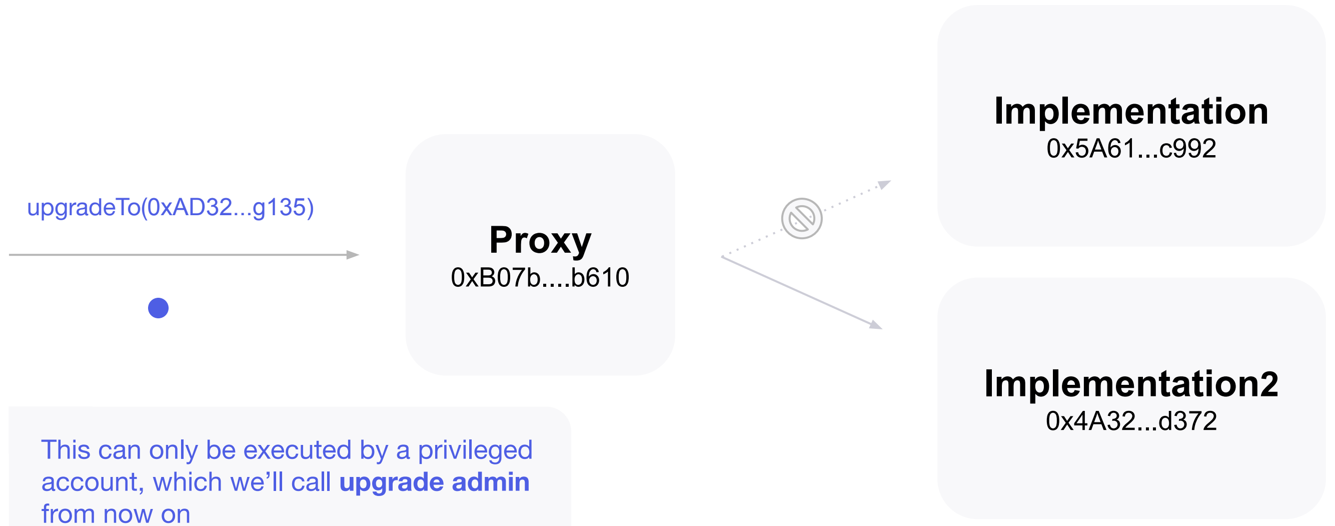


Extremely low resolution anatomy of an upgradeable contract



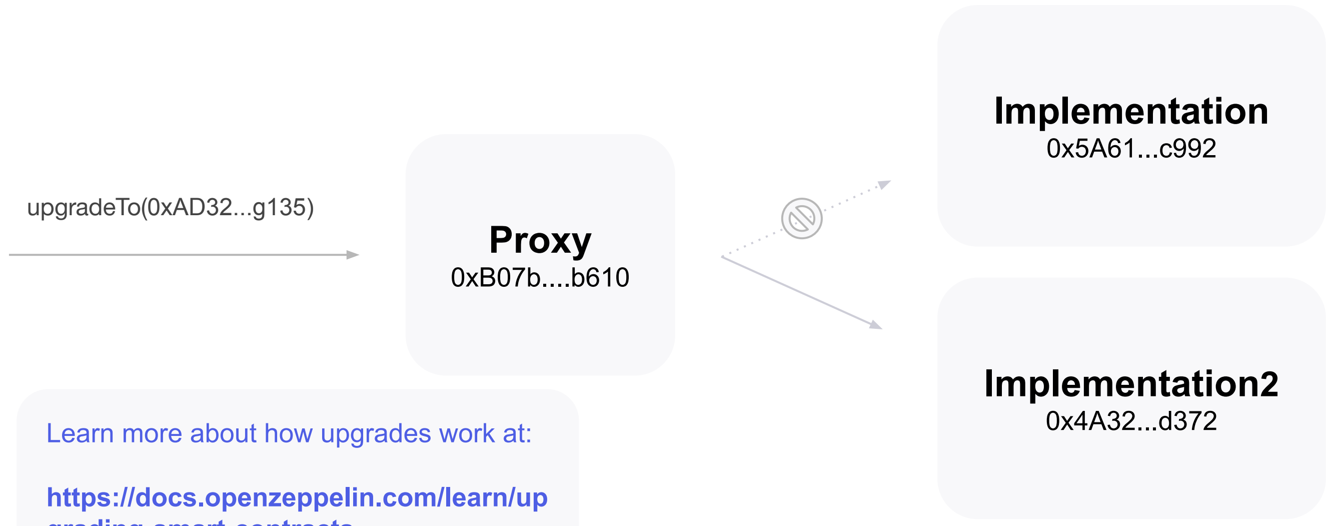
This is what happens when we **upgrade**

Extremely low resolution anatomy of an upgradeable contract



This can only be executed by a privileged account, which we'll call **upgrade admin** from now on

Extremely low resolution anatomy of an upgradeable contract



Learn more about how upgrades work at:

https://docs.openzeppelin.com/learn/upgrading-smart-contracts

Exercise

1. **Write and deploy** a simple upgradeable contract.

2. **Upgrade to a second version** of the contract from the CLI, using our dev account.

Prerequisites

- A **Defender account**
- **ETH** to pay for gas (we'll use Rinkeby for this workshop)
- An empty **Hardhat Project**, with **Hardhat Upgrades Plugin**, **Hardhat Defender Plugin**, **ethers.js**, and **dotenv.** *
- Follow along by checking out: https://github.com/OpenZeppelin/workshops/tree/master/05-upgrades-management/code

* *You can find all these details in the Defender upgrades guide:* https://docs.openzeppelin.com/defender/guide-upgrades

V1 of our contract

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

contract Box is Initializable {
    uint256 private value;

    // Emitted when the stored value changes
    event ValueChanged(uint256 newValue);

    function initialize(uint256 initialValue) public initializer {
        value = initialValue;
    }

    // Stores a new value in the contract
    function store(uint256 newValue) public {
        value = newValue;
        emit ValueChanged(newValue);
    }

    // Reads the last stored value
    function retrieve() public view returns (uint256) {
        return value;
    }

    function version() public virtual pure returns (string memory) {
        return "1.0.0";
    }
}
```

OpenZeppelin

https://openzeppelin.com

V1 of our contract

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

contract Box is Initializable {
    uint256 private value;

    // Emitted when the stored value changes
    event ValueChanged(uint256 newValue);

    function initialize(uint256 initialValue) public initializer {
        value = initialValue;
    }

    // Stores a new value in the contract
    function store(uint256 newValue) public {
        value = newValue;
        emit ValueChanged(newValue);
    }

    // Reads the last stored value
    function retrieve() public view returns (uint256) {
        return value;
    }

    function version() public virtual pure returns (string memory) {
        return "1.0.0";
    }
}
```

You can't have or call constructors in upgradeable contracts!

Read more on restrictions when writing upgradeable contracts at:

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

V1 of our contract

You cannot change storage layout in future versions!

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

contract Box is Initializable {
    uint256 private value;

    // Emitted when the stored value changes
    event ValueChanged(uint256 newValue);

    function initialize(uint256 initialValue) public initializer {
        value = initialValue;
    }

    // Stores a new value in the contract
    function store(uint256 newValue) public {
        value = newValue;
        emit ValueChanged(newValue);
    }

    // Reads the last stored value
    function retrieve() public view returns (uint256) {
        return value;
    }

    function version() public virtual pure returns (string memory) {
        return "1.0.0";
    }
}
```

Read more on restrictions when writing upgradeable contracts at:

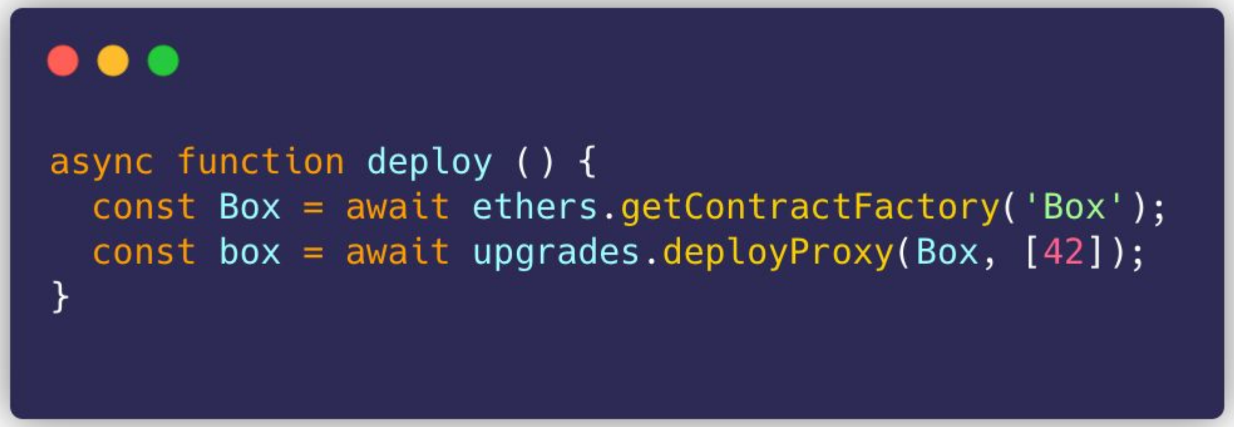
https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable

OpenZeppelin Upgrade plugins

- Integrate upgrades into your existing workflow
- Available for **Truffle** and **Hardhat**
- **Deploy** upgradeable contracts
- **Upgrade** deployed contracts
- **Manage** admin rights
- Automated sanity checks of all operations

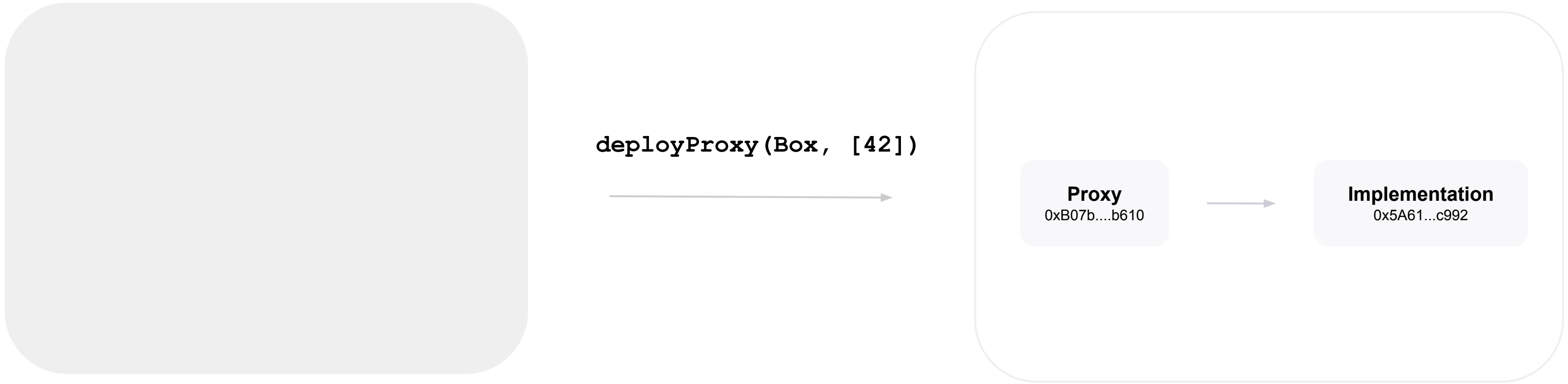
Learn more: https://docs.openzeppelin.com/upgrades-plugins/1.x/

Deploy V1 behind a proxy

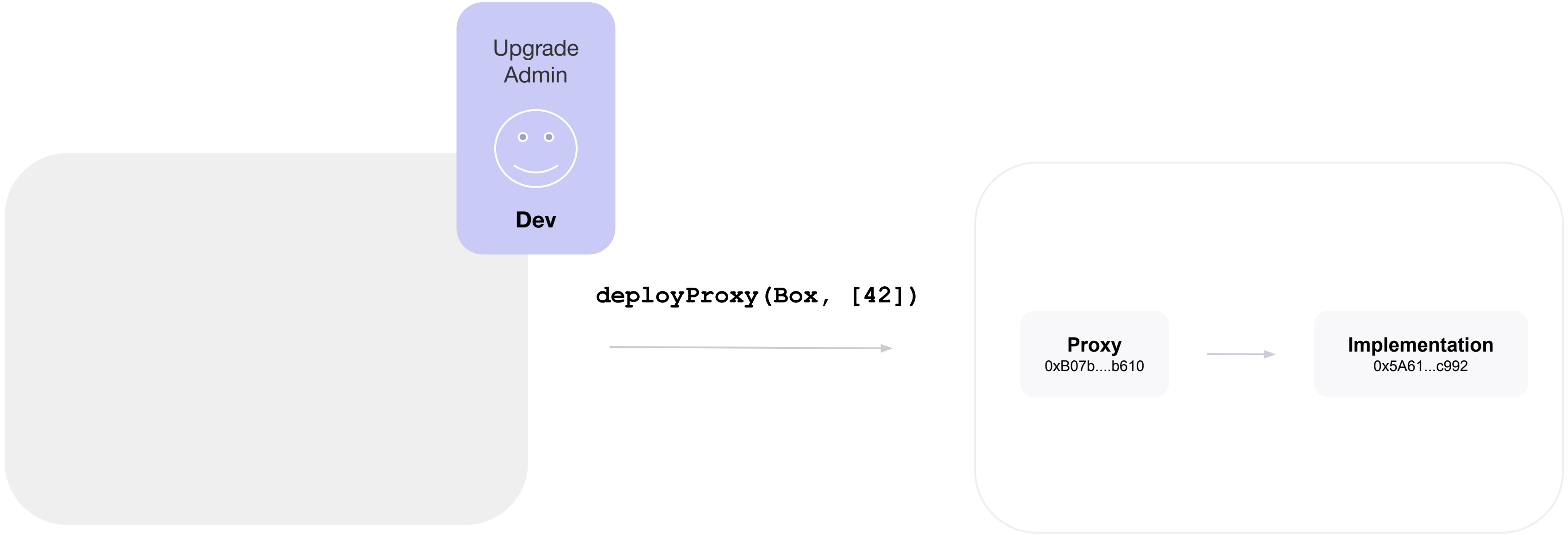


```
async function deploy () {
  const Box = await ethers.getContractFactory('Box');
  const box = await upgrades.deployProxy(Box, [42]);
}
```

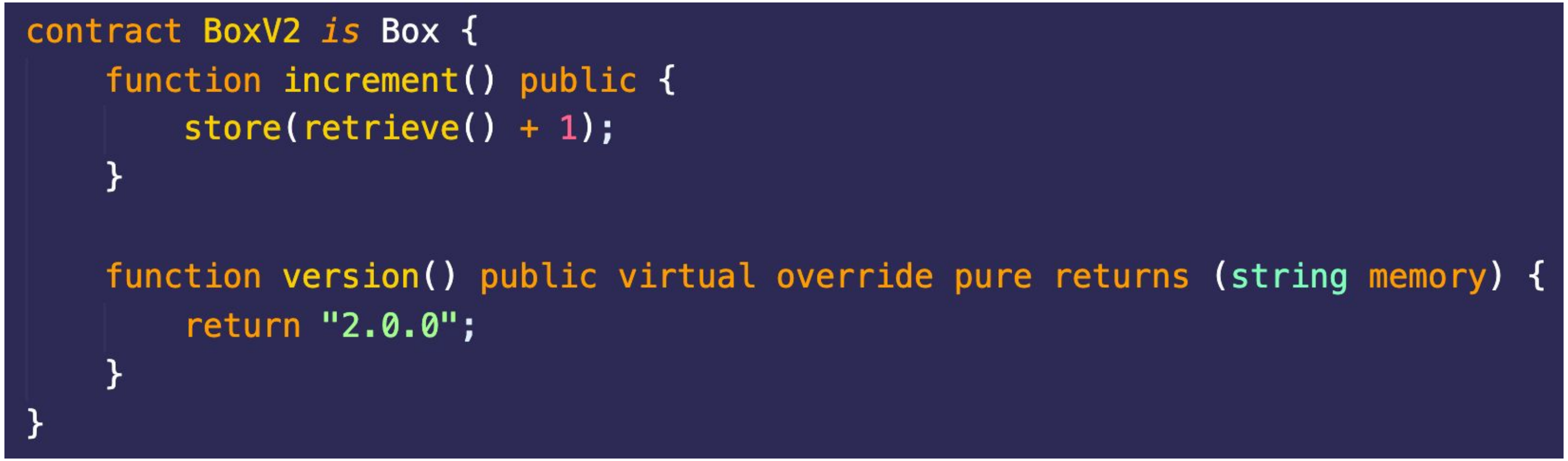
Deploy V1 behind a proxy



Deploy V1 behind a proxy: Upgrades admin



V2 of Box



```
contract BoxV2 is Box {
    function increment() public {
        store(retrieve() + 1);
    }

    function version() public virtual override pure returns (string memory) {
        return "2.0.0";
    }
}
```

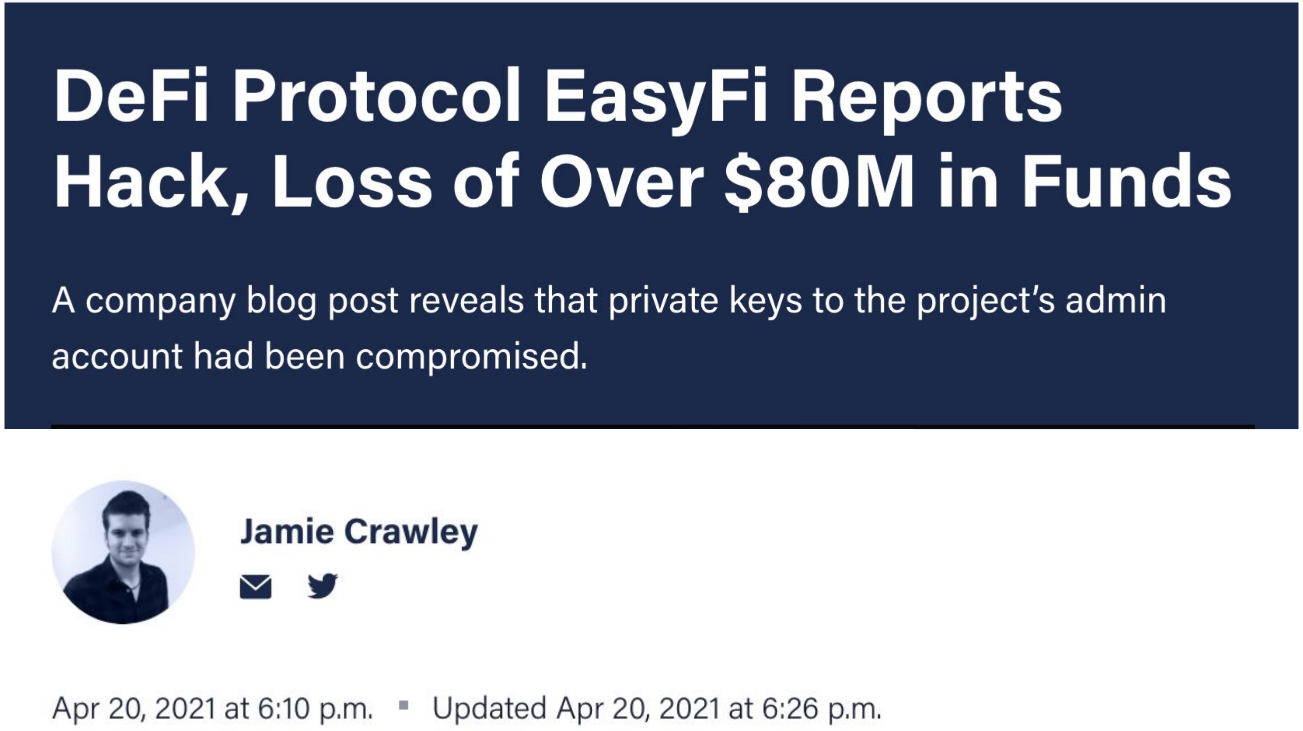
Let’s require less trust. Multisigs!

We’ll transfer upgrade admin rights to a **Multisig**, so that we:

- Mitigate impact of **lost or compromised individual keys**
- **Distribute and dilute authority** of a single account holder by involving many
- Ensure all relevant groups of **stakeholders are represented**

Learn more with Defender Advisor!
https://defender.openzeppelin.com/#/advisor/docs/use-multiple-signatures-for-critical-administrative-tasks

Let’s require less trust. Multisigs!


DeFi Protocol EasyFi Reports Hack, Loss of Over $80M in Funds

A company blog post reveals that private keys to the project’s admin account had been compromised.

Jamie Crawley

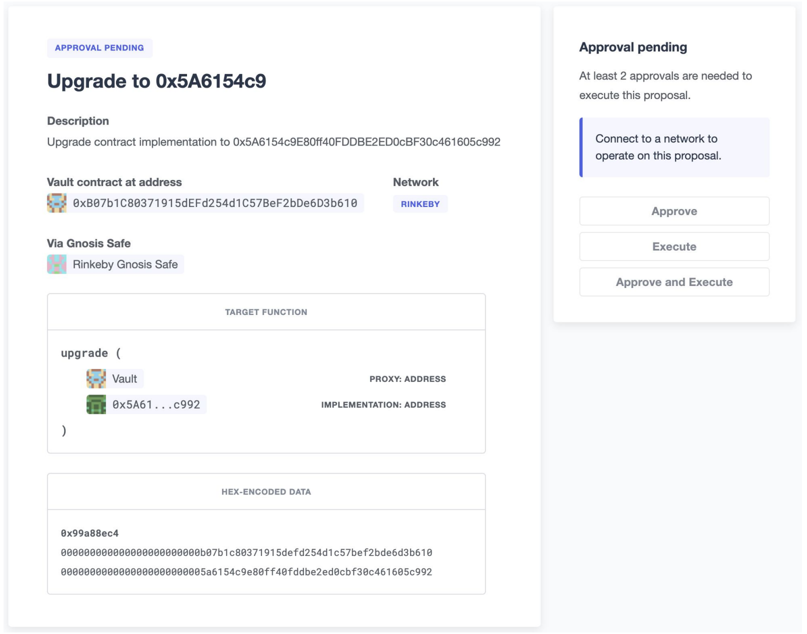
Apr 20, 2021 at 6:10 p.m. ▪ Updated Apr 20, 2021 at 6:26 p.m.

Defender Admin - Automate and secure all your smart contract administration

Use **multi-sigs** to administrate your contract to:

- Tweak critical parameters
- **Pause** in the event of an emergency
- **Upgrade** your contract to a new implementation

- No need for privileged access for Defender
- Simple UI
- Trigger workflows via API



APPROVAL PENDING

Upgrade to 0x5A6154c9

Description
Upgrade contract implementation to 0x5A6154c9E80ff40FDDBE2ED0cBF30c461605c992

Vault contract at address
0xB07b1C80371915dEFd254d1C57BeF2bDe6D3b610

Network
RINKEBY

Via Gnosis Safe
Rinkeby Gnosis Safe

TARGET FUNCTION

```
upgrade (
    Vault                 PROXY: ADDRESS
    0x5A61...c992         IMPLEMENTATION: ADDRESS
)
```

HEX-ENCODED DATA

```
0x99a88ec4
000000000000000000000000b07b1c80371915defd254d1c57bef2bde6d3b610
0000000000000000000000005a6154c9e80ff40fddbe2ed0cbf30c461605c992
```

Approval pending

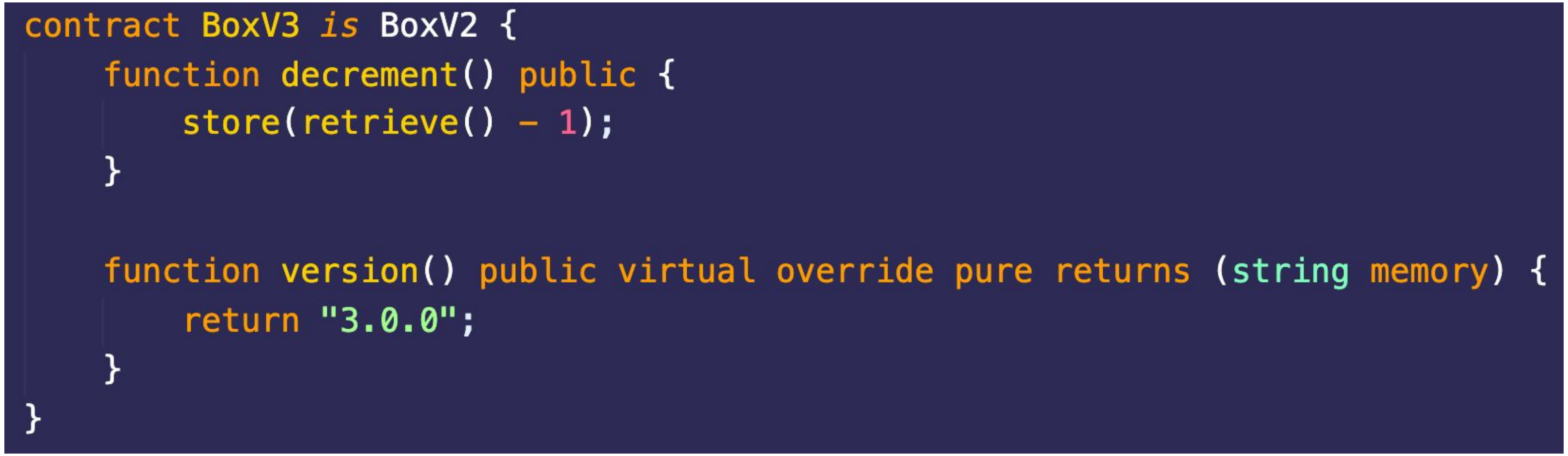
At least 2 approvals are needed to execute this proposal.

Connect to a network to operate on this proposal.

Exercise 2: same process, but with a multisig and Defender Admin API

1. **Write** a third version of our contract.
2. **Transfer** upgrade rights to a Gnosis Safe Multisig.
3. **Create** a Defender Admin Proposal to upgrade.
4. **Use Defender Admin** to approve and monitor execution of the proposal.

V3 of Box



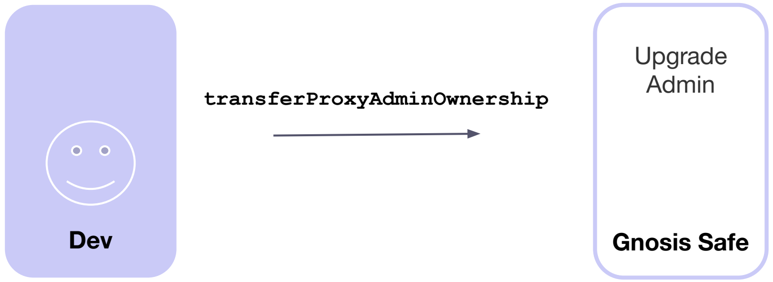
```
contract BoxV3 is BoxV2 {
    function decrement() public {
        store(retrieve() - 1);
    }

    function version() public virtual override pure returns (string memory) {
        return "3.0.0";
    }
}
```

Transfer upgrade rights to a Gnosis Safe Multisig

Upgrade Admin
Dev
Gnosis Safe

Transfer upgrade rights to a Gnosis Safe Multisig



Recap

1. **Wrote and deployed** Box V1, with help from OZ Upgrades Plugins.
2. **Wrote, deployed** and upgraded to Box V2.
3. **Transferred upgrade** admin powers from a single dev account to a multisig.
4. **Wrote**, deployed and proposed an upgrade to Box V3, with Upgrades Plugins + Defender Admin API.
5. **Collectively reviewed**, approved and executed the upgrade proposal via the multisig, with Defender Admin UI.

Coming soon to Defender...

1. **Timelocks**: give your users opt-out guarantees
2. **Batched transactions**: trigger multiple orchestrated transactions with a single Defender Admin Proposal.
3. **Granular access level control**: compartmentalize power.
4. **Public proposals**: be more transparent with your community.
5. **Snapshot integration**: link community/stakeholder votes with specific Defender Admin Proposals.

Learn more

- OpenZeppelin Upgrade Guides:
  https://docs.openzeppelin.com/learn/upgrading-smart-contracts

- Managing Upgrades through Defender:
  https://docs.openzeppelin.com/defender/guide-upgrades

- Defender Advisor article on multisigs:
  https://defender.openzeppelin.com/#/advisor/docs/use-multiple-signatures-for-critical-administrative-tasks

- Code for this workshop:
  https://github.com/OpenZeppelin/workshops/tree/master/05-upgrades-management/code

Q&A