OpenZeppelin

Community Call #2

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.

We're hiring!

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.

Community Call Goals

- Give visibility to new features & improvements
- Involve you in the process
  - Provide feedback
  - Request features
  - Engage discussion
  - Review the code

Contracts

@openzeppelin/contracts@4.5.0-rc.0

@openzeppelin/contracts-upgradable@4.5.0-rc.0

OpenZeppelin Contracts Security Process

- Internal code reviews
- 99% test coverage
- External audit and formal verification
- Community review
  - Release candidates
  - Open Review Period
  - Bug Bounty on Immunefi

449 lines (339 sloc) | 43.8 KB

Raw Blame

Changelog

4.5.0

- `ERC2891` : add implementation of the royalty standard, and the respective extensions for `ERC721` and `ERC1155` . (#3012)
- `GovernorTimelockControl` : improve the `state()` function to have it reflect cases where a proposal has been canceled directly on the timelock. (#2977)
- `Math` : add a `abs(int256)` method that returns the unsigned absolute value of a signed value. (#2984)
- Preset contracts are now deprecated in favor of Contracts Wizard. (#2986)
- `Governor` : add a relay function to help recover assets sent to a governor that is not its own executor (e.g. when using a timelock). (#2926)
- `GovernorPreventLateQuorum` : add new module to ensure a minimum voting duration is available after the quorum is reached. (#2973)
- `ERC721` : improved revert reason when transferring from wrong owner. (#2975)
- `Votes` : Added a base contract for vote tracking with delegation. (#2944)
- `ERC721Votes` : Added an extension of ERC721 enabled with vote tracking and delegation. (#2944)
- `ERC2771Context` : use immutable storage to store the forwarder address, no longer an issue since Solidity >=0.8.8 allows reading immutable variables in the constructor. (#2917)
- `Base64` : add a library to parse bytes into base64 strings using `encode(bytes memory)` function, and provide examples to show how to use to build URL-safe `tokenURIs` . (#2884)
- `ERC20` : reduce allowance before triggering transfer. (#3056)
- `ERC20` : do not update allowance on `transferFrom` when allowance is `type(uint256).max` . (#3085)
- `ERC777` : do not update allowance on `transferfrom` when allowance is `type(uint256).max` . (#3085)
- `SignedMath` : a new signed version of the Math library with `max` , `min` , and `average` . (#2686)
- `ERC1967Upgrade` : Refactor the secure upgrade to use `ERC1822` instead of the previous rollback mechanism. This reduces code complexity and attack surface with similar security guarantees. (#3021)
- `UUPSUpgradeable` : Add `ERC1822` compliance to support the updated secure upgrade mechanism. (#3021)
- Some more functions have been made virtual to customize them via overrides. In many cases this will not imply that other functions in the contract will automatically adapt to the overridden definitions. People who wish to override should consult the source code to understand the impact and if they need to override any additional functions to achieve the desired behavior.

Breaking changes

- `ERC1967Upgrade` : The function `_upgradeToAndCallSecure` was renamed to `_upgradeToAndCallUUPS` , along with the change in security mechanism described above.
- `Address` : The Solidity pragma is increased from `^0.8.0` to `^0.8.1` . This is required by the `account.code.length` syntax that replaces inline assembly. This may require users to bump their compiler version from `0.8.0` to `0.8.1` or later. Note that other parts of the code already include stricter requirements.

UUPSUpgradeable simplified

Removed “rollback test”.

Goal: Simplify code to reduce attack surface.

Remain protected against accidents.

Pending proposal to extend ERC-1822.

```
try IERC1822Proxiable(newImplementation).proxiableUUID() returns (bytes32 slot) {
    require(slot == _IMPLEMENTATION_SLOT, "ERC1967Upgrade: unsupported proxiableUUID");
} catch {
    revert("ERC1967Upgrade: new implementation is not UUPS");
}
_upgradeToAndCall(newImplementation, data, forceCall);
```

37 contracts/proxy/ERC1967/ERC1967Upgrade.sol

```
-       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);
+       // Upgrades from old implementations will perform a rollback test. This test requires the new
+       // implementation to upgrade back to the old, non-ERC1822 compliant, implementation. Removing
+       // this special case will break upgrade paths from old UUPS implementation to new ones.
+       if (StorageSlot.getBooleanSlot(_ROLLBACK_SLOT).value) {
+           _setImplementation(newImplementation);
+       } else {
+           try IERC1822Proxiable(newImplementation).proxiableUUID() returns (bytes32 slot) {
+               require(slot == _IMPLEMENTATION_SLOT, "ERC1967Upgrade: unsupported proxiableUUID");
+           } catch {
+               revert("ERC1967Upgrade: new implementation is not UUPS");
+           }
+           _upgradeToAndCall(newImplementation, data, forceCall);
```

Votes & ERC721Votes

Goal: Enable governance protocols with NFT-based voting. (1 NFT = 1 vote)

Modularize vote-tracking logic.

```
interface IVotes {
    event DelegateChanged(address indexed delegator, address indexed
    event DelegateVotesChanged(address indexed delegate, uint256 prev
    function getVotes(address account) external view returns (uint256
    function getPastVotes(address account, uint256 blockNumber) exter
    function getPastTotalSupply(uint256 blockNumber) external view re
    function delegates(address account) external view returns (addres
    function delegate(address delegatee) external;
    function delegateBySig(
```

```
abstract contract ERC721Votes is ERC721, Votes {
    /**
     * @dev Adjusts votes when tokens are transferred.
     *
     * Emits a {Votes-DelegateVotesChanged} event.
     */
    function _afterTokenTransfer(
        address from,
        address to,
        uint256 tokenId
    ) internal virtual override {
        _transferVotingUnits(from, to, 1);
        super._afterTokenTransfer(from, to, tokenId);
    }

    /**
     * @dev Returns the balance of `account`.
     */
    function _getVotingUnits(address account) internal virtual override returns (uint256) {
        return balanceOf(account);
    }
}
```

NFT Royalties (ERC2981)

Goal: Support ERC2981 for NFT royalties.

```
function royaltyInfo(uint256 _tokenId, uint256 _salePrice) external view override returns (address, uint256) {
    RoyaltyInfo memory royalty = _tokenRoyaltyInfo[_tokenId];

    if (royalty.receiver == address(0)) {
        royalty = _defaultRoyaltyInfo;
    }

    uint256 royaltyAmount = (_salePrice * royalty.royaltyFraction) / _feeDenominator();

    return (royalty.receiver, royaltyAmount);
}
```

Gas optimization: infinite allowance for ERC20 & ERC777

Goal: Reduce gas cost of some transferFrom calls.

PR proposed by @[0xclaudeshannon]

```
    function transferFrom(
        address sender,
        address recipient,
        uint256 amount
    ) public virtual override returns (bool) {
        uint256 currentAllowance = _allowances[sender][_msgSender()];
+       if (currentAllowance ≠ type(uint256).max) {
            require(currentAllowance ≥ amount, "ERC20: transfer amount exceeds allowance");
            unchecked {
                _approve(sender, _msgSender(), currentAllowance - amount);
            }
+       }

        _transfer(sender, recipient, amount);

        return true;
    }
```

SignedMath

Goal: Support math operations on int256.

PR proposed by @rotcivegaf and @barakman

```
library SignedMath {
    /**
     * @dev Returns the largest of two signed numbers.
     */
    function max(int256 a, int256 b) internal pure returns (int256) {
        return a ≥ b ? a : b;
    }

    /**
     * @dev Returns the smallest of two signed numbers.
     */
    function min(int256 a, int256 b) internal pure returns (int256) {
        return a < b ? a : b;
    }
```

```
/**
 * @dev Returns the average of two signed numbers without overflow.
 * The result is rounded towards zero.
 */
function average(int256 a, int256 b) internal pure returns (int256) {
    // Formula from the book "Hacker's Delight"
    int256 x = (a & b) + ((a ^ b) >> 1);
    return x + (int256(uint256(x) >> 255) & (a ^ b));
}

/**
 * @dev Returns the absolute unsigned value of a signed value.
 */
function abs(int256 n) internal pure returns (uint256) {
    unchecked {
        // must be unchecked in order to support `n = type(int256).min`
        return uint256(n ≥ 0 ? n : -n);
    }
}
```

Base64

Goal: Bytes to base64 string conversion.

PR proposed by [@ernestognw]

```
8   library Base64 {
9       /**
10       * @dev Base64 Encoding/Decoding Table
11       */
12      string internal constant _TABLE = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
13
14      /**
15       * @dev Converts a `bytes` to its Bytes64 `string` representation.
16       */
17      function encode(bytes memory data) internal pure returns (string memory) {
18          /**
19           * Inspired by Brecht Devos (Brechtpd) implementation - MIT licence
20           * https://github.com/Brechtpd/base64/blob/e78d9fd951e7b0977ddca77d92dc85183770daf4/base64.sol
21           */
22          if (data.length == 0) return "";
23
24          // Loads the table into memory
25          string memory table = _TABLE;
26
27          // Encoding takes 3 bytes chunks of binary data from `bytes` data parameter
28          // and split into 4 numbers of 6 bits.
29          // The final Base64 length should be `bytes` data length multiplied by 4/3 rounded up
30          //   - `data.length + 2`  -> Round up
31          //   - `/ 3`              -> Number of 3-bytes chunks
32          //   - `4 *`              -> 4 characters for each chunk
33          string memory result = new string(4 * ((data.length + 2) / 3));
34
35          assembly {
36              // Prepare the lookup table (skip the first "length" byte)
37              let tablePtr := add(table, 1)
38
39              // Prepare result pointer, jump over length
40              let resultPtr := add(result, 32)
```

Contribute

Participate in our bug bounty on Immunefi.

Rewards of up to $25,000 and a special POAP for each release.

Open Review Period

4.5 Release Candidate - Available Now!

https://zpl.in/contracts/v/4.5

Contribute

zpl.in/contracts/requests

OpenZeppelin | forum

OpenZeppelin Contracts - Feature requests [Q1 2022]

General

frangio OpenZeppelin Team 184 1m

Please use this thread to share the contracts or utilities that you'd most like to see added to OpenZeppelin Contracts.

Share one idea per answer and use the like button to vote for other suggestions!

An area we're actively looking into is Layer 2 and bridges, so if you have specific requests around this topic they're specially welcome!

Upgrades plugins

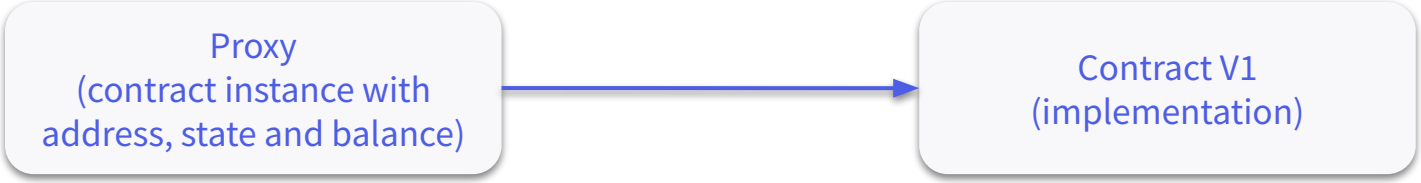
@openzeppelin/hardhat-upgrades@1.13.0

@openzeppelin/truffle-upgrades@1.12.0

Upgradable contracts

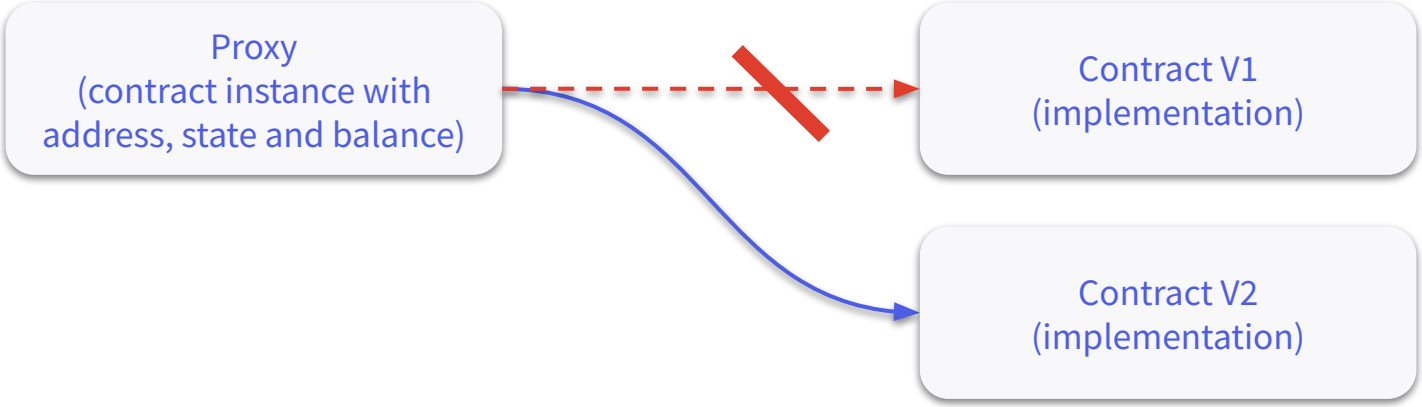
- Smart contracts are immutable by default
- In some scenarios, you may want contracts that can be changed
  - Bug fixes, security fixes
  - Code improvements, new features
- OpenZeppelin Contracts provide three proxy patterns for upgradability: UUPS, transparent, and beacon proxies

UUPS and transparent proxies



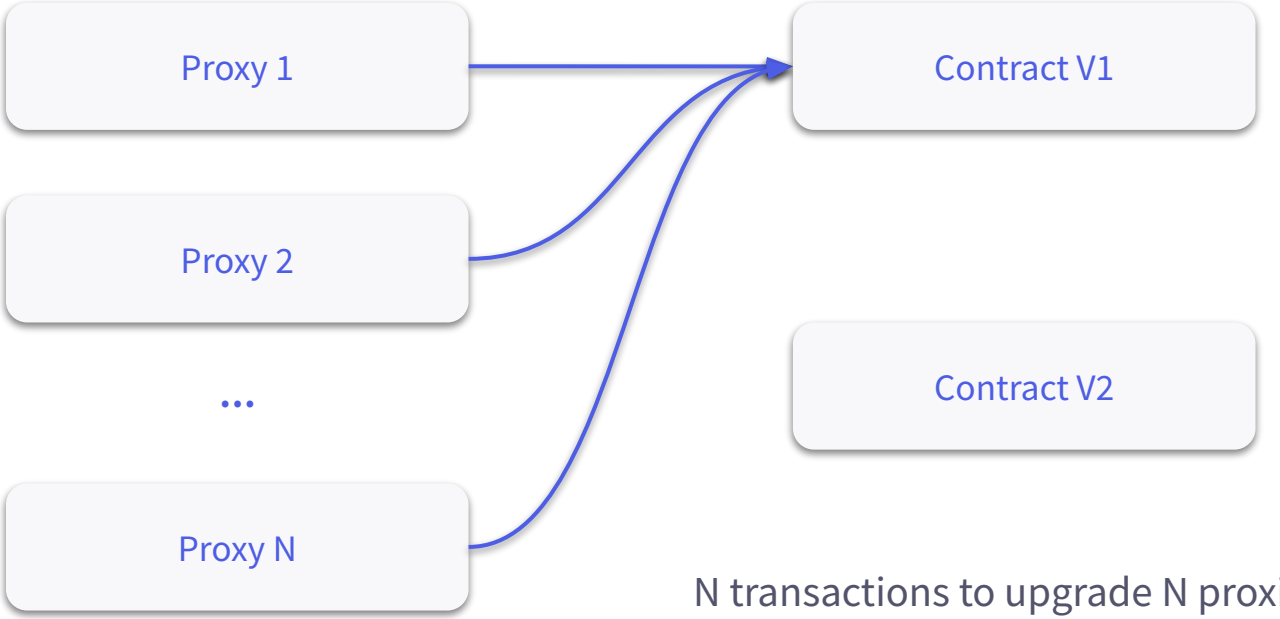
Proxies *delegate* calls to their implementation

Upgrading proxies



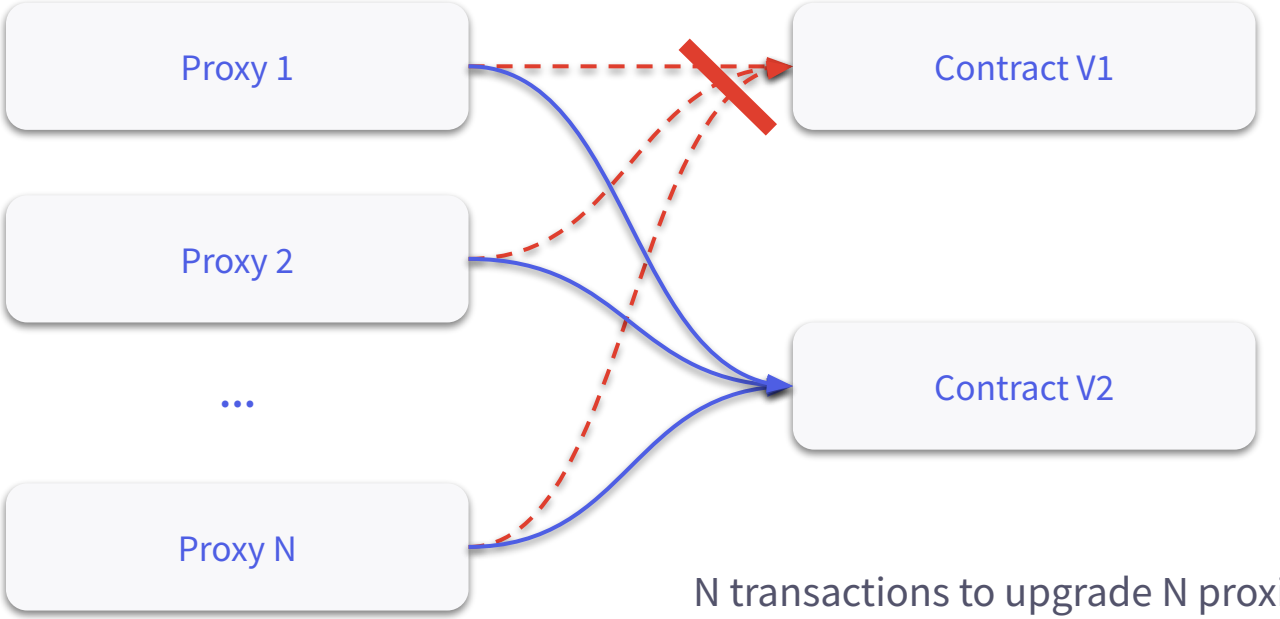
Upgrading a proxy changes its implementation while preserving address, state, and balance

Upgrading multiple proxies



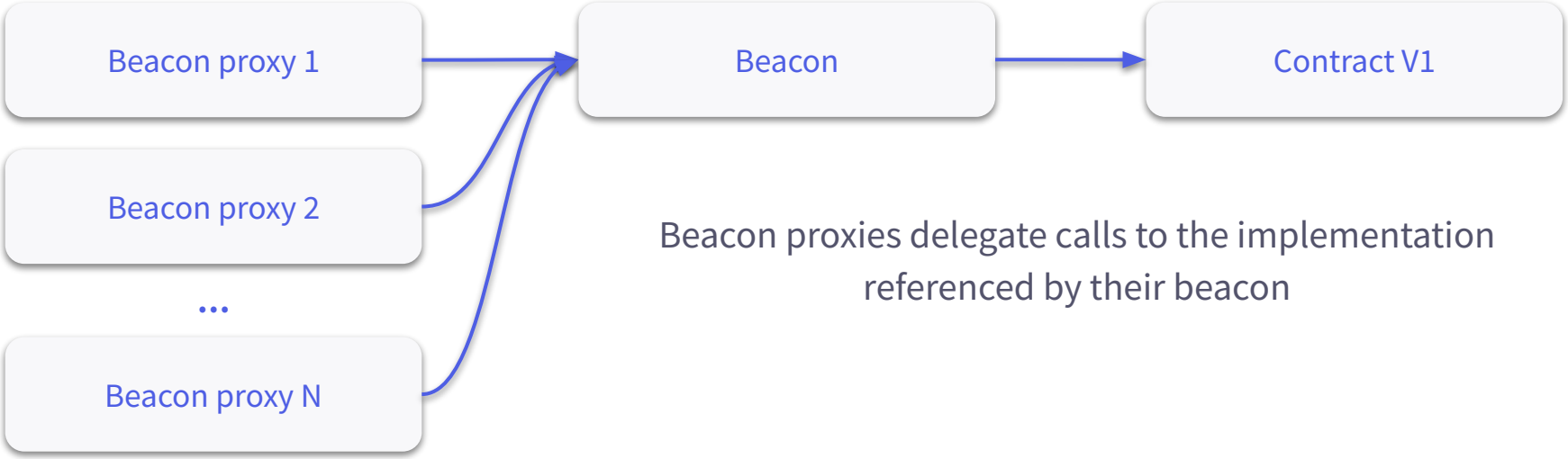
N transactions to upgrade N proxies

Upgrading multiple proxies



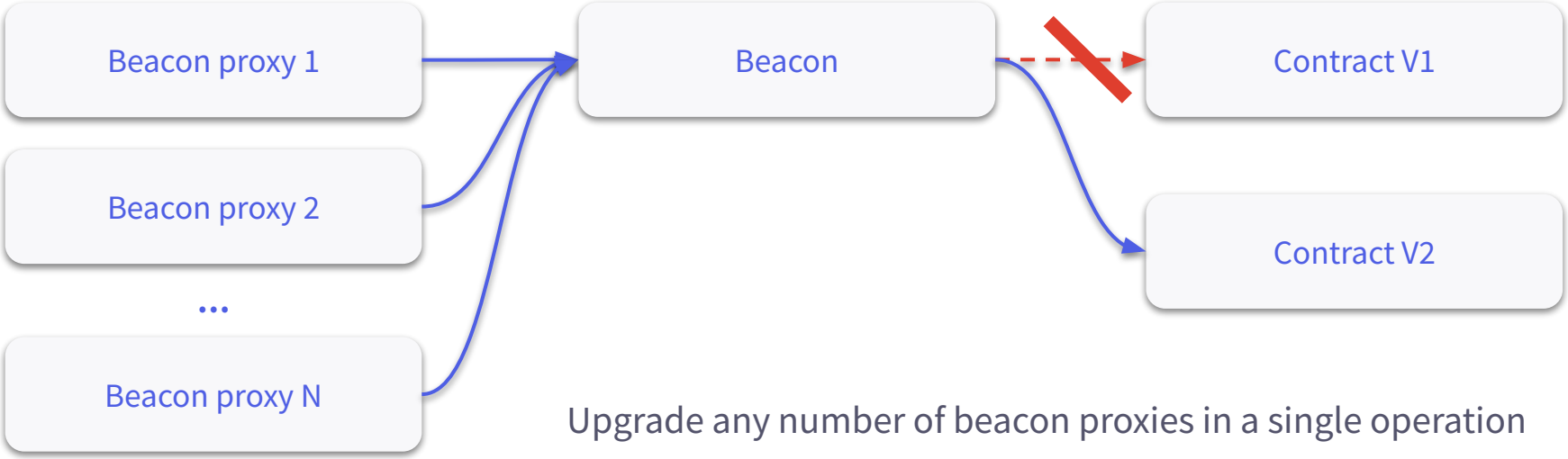
N transactions to upgrade N proxies

Beacon proxies



Beacon proxies delegate calls to the implementation referenced by their beacon

Beacon proxies



Upgrade any number of beacon proxies in a single operation

Upgrades plugins

- Deploy and manage upgradable contracts in Hardhat or Truffle
- Validates that your contracts are **upgrade safe**

Deploying and managing proxies

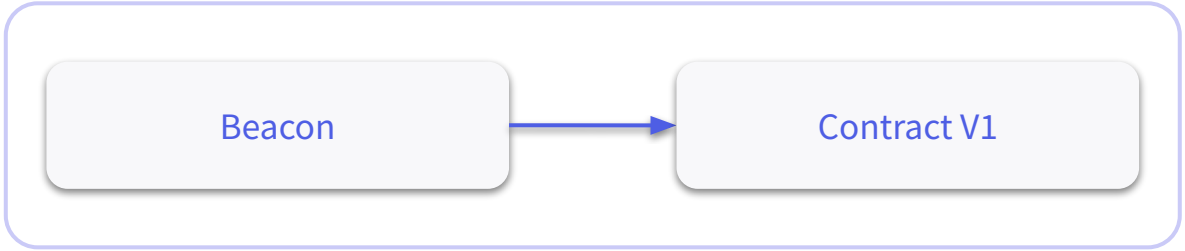
- UUPS and transparent proxies

  `deployProxy(), upgradeProxy()`

- **New:** Beacon proxies

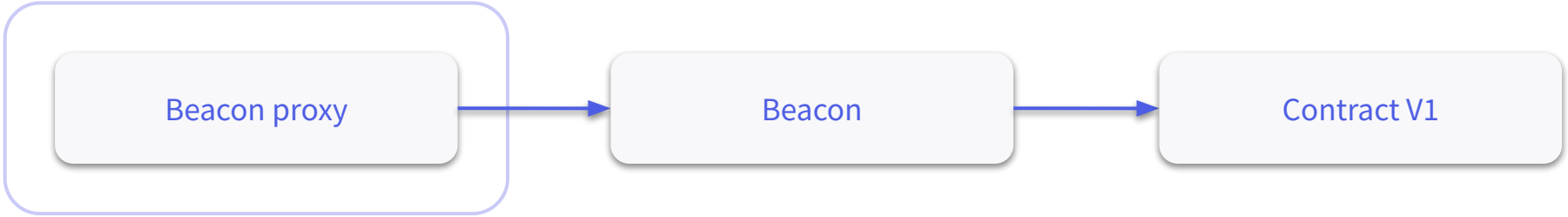
  `deployBeacon(), deployBeaconProxy(), upgradeBeacon()`

Deploying a beacon



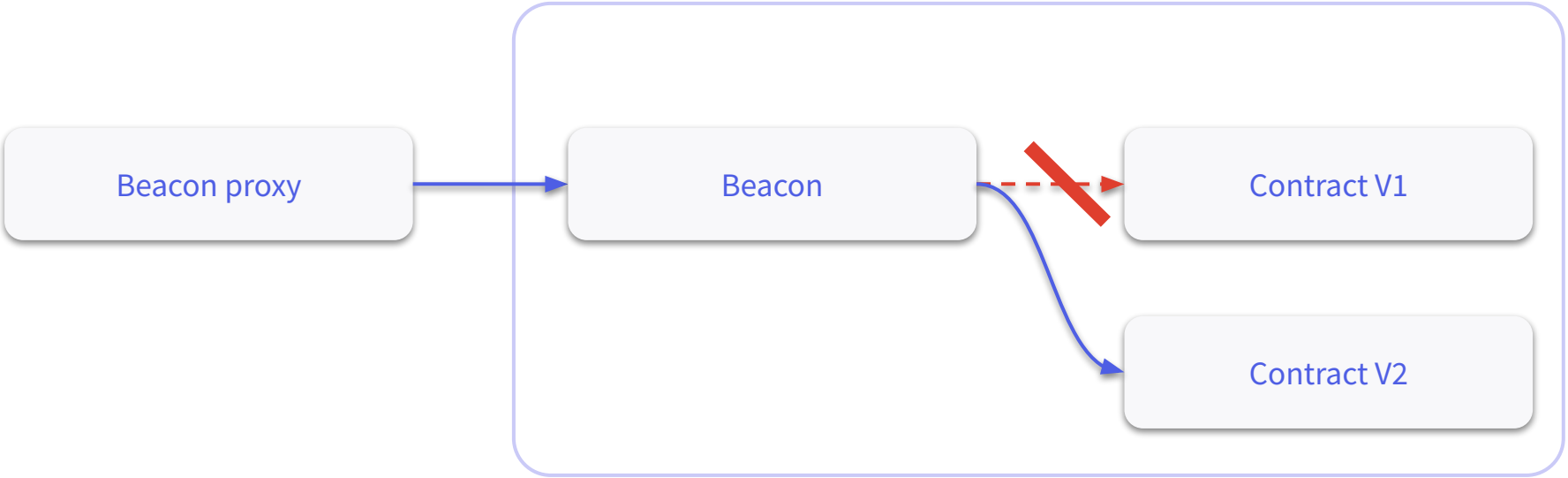

```
deployBeacon(ContractV1);
```

Deploying a beacon proxy




```
deployBeaconProxy(beacon,
ContractV1, ["arg for initializer"]);
```

Upgrading a beacon




```
upgradeBeacon(beacon, ContractV2);
```

Learn more

docs.openzeppelin.com/upgrades-plugins

← Home

Upgrades Plugins

- Overview
- Using with Truffle
- Using with Hardhat
- Writing Upgradeable Contracts
- Proxy Upgrade Pattern
- Network Files
- Migrate from CLI
- Frequently Asked Questions
- API Reference
  - Truffle Upgrades
  - Hardhat Upgrades
  - Core

OpenZeppelin | docs

GitHub Forum Blog Website

Upgrades Plugins

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

- Deploy upgradeable contracts.
- Upgrade deployed contracts.
- Manage proxy admin rights.
- Easily use in tests.

TIP

Upgrades Plugins are only a part of a comprehensive set of OpenZeppelin tools for deploying and securing upgradeable smart contracts. Check out the full list of resources.

UPGRADES PLUGINS

- Overview
  - Installation
  - Usage
- How the plugins work
- Proxy patterns
- Managing ownership

Overview

Defender

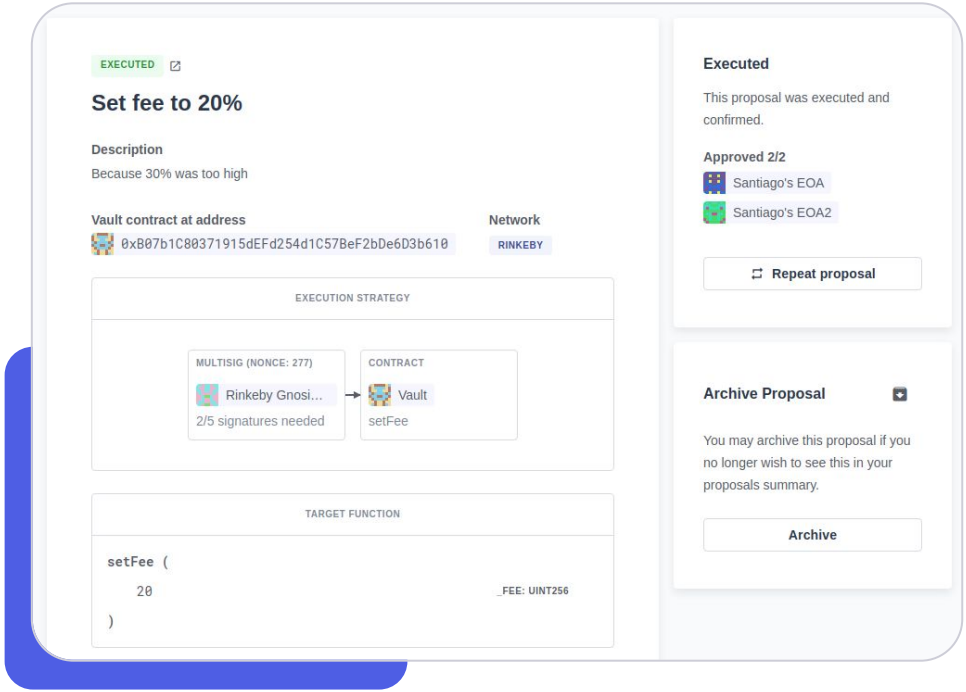
Secure operations platform for smart contract systems

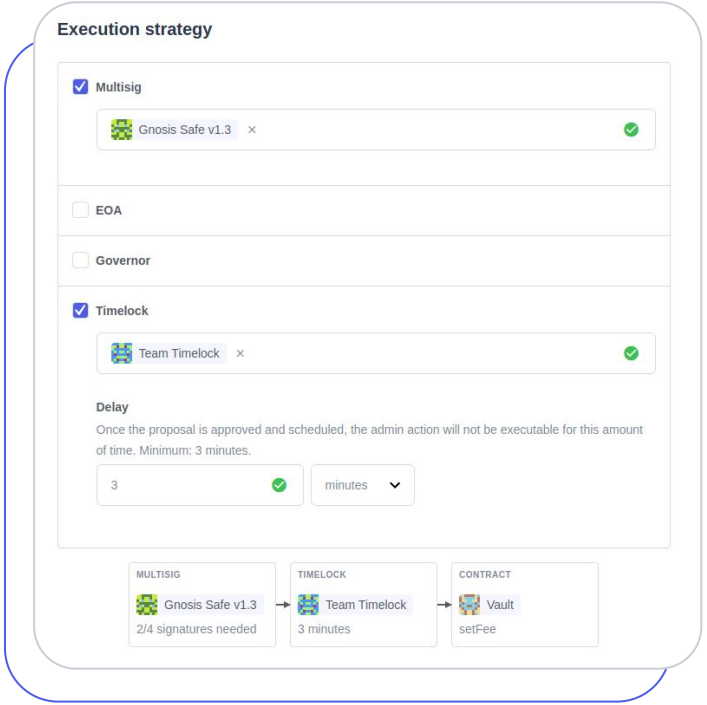
zpl.in/**defender**

Security, Reliability and Risk Management

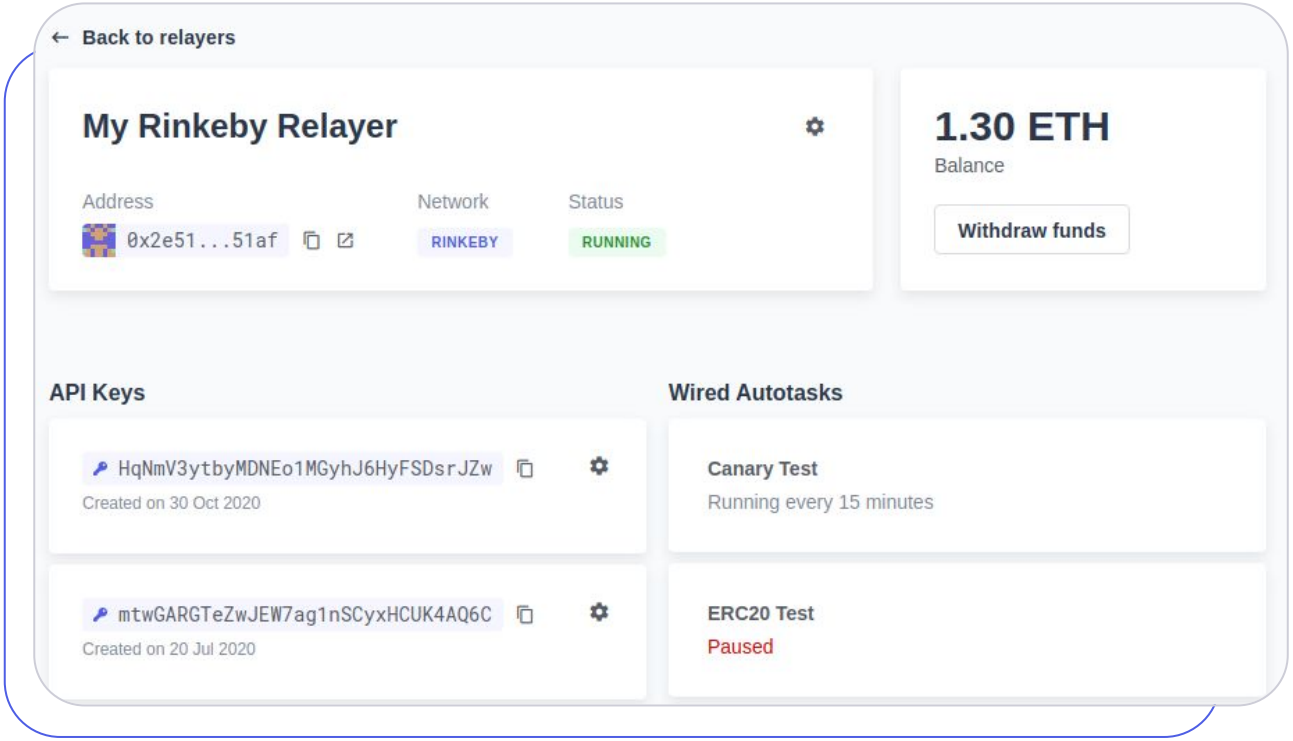
OpenZeppelin provides a complete suite of **security and reliability products** to build, manage, and inspect all aspects of software development and operations for Ethereum projects.

Admin / Manage contracts via Multisigs, Timelocks, or Governance





Relayers / Secure vaults for private keys and simplified txs



Sentinels / Monitor contracts

← Back to sentinels

Deposits on Vault

Monitoring	Network	Status
Vault	RINKEBY	RUNNING

Monitoring

Contract

EVENTS (1/5)

`Deposited(address,uint256,uint256)`

Notifications

Santiago
Active santiago@openzeppelin.com

defender-sentinels-demo
Active

Threshold

Press edit to specify how often you want to receive Sentinel notifications.

defender-sentinel APP 6:52 PM

⚡ **Deposits in Rinkeby Vault** was triggered by a transaction ⚡

0x87c89b8be4f672f88d79e0b217604c571a43446f0dd96d0f41c2f8bbf0d2b2bc

Inspect Transaction

Matched Rules

Address: 0xB07b1C80371915dEFd254d1C57BeF2bDe6D3b610

Event

Deposited(address,uint256,uint256)

value > 1e14 AND from == '0xF0a9ed2663311ce436347bb6f240181ff103ca16'

from: 0xf0A9eD2663311CE436347Bb6F240181FF103CA16
value: 100000000000000000
fee: 15000000000000000

Autotasks / Automate scripts execution

← Back to dashboard

Sweep Tokens

Connected to	Runs every	Status
Rinkeby 01	300 minutes	RUNNING

Manually run autotask

This won't affect the scheduled triggers.

Run autotask now

Runs history

DATE	TRIGGER	STATUS
25 June 2021, 08:34	schedule	SUCCESS
25 June 2021, 03:34	schedule	SUCCESS
24 June 2021, 22:34	schedule	SUCCESS
24 June 2021, 17:34	schedule	SUCCESS

Edit Sweep Tokens code

Tip: you can update this code programmatically with the `defender-autotask-client` using the Autotask identifier: `dc70b5d4-4ecc-40b3-b972-120041c4d7b0`

Code

```
const contractAbi = [{"inputs":[],"name":"admin","outputs":[{"internalTyp
const contractAddr = '0xB07b1C80371915dEFd254d1C57BeF2bDe6D3b610';

const { ethers } = require("ethers");
const { DefenderRelaySigner, DefenderRelayProvider } = require('defender-

// Entrypoint for the Autotask
exports.handler = async function(credentials) {
  console.log(`Using client version ${require('defender-relay-client').VE
  // Initialize default provider and defender relayer signer
  const provider = new DefenderRelayProvider(credentials);
  const signer = new DefenderRelaySigner(credentials, provider, { speed:

  // Get contract instance
  const contract = new ethers.Contract(contractAddr, contractAbi, signer)

```

Advisor / Learn security best practices

TITLE	CATEGORY	RATING	EFFORT
Test Contract Upgrades	Testing	CRITICAL	MEDIUM
Data Out of Sync	Monitoring	CRITICAL	MEDIUM
Privileged Administrator Transactions	Monitoring	CRITICAL	MEDIUM
Spikes in Account Activity	Monitoring	CRITICAL	LARGE
Implement Reentrancy Protections	Development	CRITICAL	SMALL
Recast Variables Safely	Development	CRITICAL	SMALL
Assert Revert Reasons	Testing	CRITICAL	SMALL
Drop In System Funds	Monitoring	CRITICAL	MEDIUM
Post-Mortem Analysis	Operations	CRITICAL	MEDIUM
Access Arrays Using Enumeration and Pagination	Development	CRITICAL	MEDIUM
Prevent Replay Attacks When Using Signatures	Development	CRITICAL	MEDIUM
Secure All Administrative Keys	Operations	CRITICAL	MEDIUM
Test Emission of Events	Testing	HIGH	SMALL
Collateral Ratios	Monitoring	HIGH	MEDIUM

Test Contract Upgrades

- Applies To: upgrades

Before upgrading a live mainnet smart contract to a new implementation, it's critical to test the upgrade process itself, even if both the old and new implementations are correct, since the upgrade itself can potentially introduce issues.

Description

Smart contract upgrades allow changing the code executed in a contract, while preserving the existing contract state, balance, and address. Regardless of the upgrade pattern being used, it's important to test the upgrade before actually executing it on mainnet.

Tests should verify not only the behavior of the upgraded implementation, but also that state was correctly preserved during the upgrade, and that it's possible to rollback if needed. Tests should be run on local development nodes, testnets, and ideally on mainnet forks as well. It's also a good idea to include static analyzers if available for the upgrade pattern being used.

What to test

A good setup for testing starts by deploying and seeding the original version of the contract, and then executes the upgrade using the same pattern as the live contract uses. Tests should assert that the new implementation behaves correctly, and that all state and balance has been properly preserved. Note that testing the new implementation in isolation is not enough, and any test suite developed for it should be re-run on an upgraded instance as well. For example, in a delegate-call proxy upgrade pattern, a new implementation that redefines the order of storge variables may corrupt the contract state when upgrading; this cannot be detected by testing the old or new implementations alone, and can only be found by performing an upgrade.

Additionally, if there are any migration methods to be executed during the upgrade, such as calculating the value for a new field, these should be tested as well. Furthermore, it's important to test that these methods, after being called for the migration, cannot be called again

Sentinels API

Manage your Sentinels programmatically

Sentinel API - What are Sentinels?

- **Contract** Sentinels
  - Monitoring of transactions to a contract
  - Conditions defined by events, functions, transaction parameters
- **Forta** Sentinels
  - Monitoring of Forta Alerts
  - Conditions defined by agents, contract address, alert IDs, severity
- Notification via email, slack, telegram, discord or **autotask** execution

Sentinel API - Benefits

- Add sentinel creation to deployment script of a smart contract
- Bulk manage sentinels
- Pause sentinels based on custom conditions

Sentinel API - Resources

Example script

https://github.com/OpenZeppelin/defender-client/blob/master/examples/create-sentinel/index.js

Package

https://github.com/OpenZeppelin/defender-client/tree/master/packages/sentinel

Documentation

https://docs.openzeppelin.com/defender/sentinel-api-reference

Forta Sentinels

Monitor Forta Agents from your Defender Sentinels

zpl.in/**docs**
zpl.in/**blog**
zpl.in/**forum**
zpl.in/**defender**
zpl.in/**events**
zpl.in/**join**

We're hiring!

zpl.in/**join**

Blockchain Security Engineer
Full Stack Ethereum Developer
Application UX Designer
Technical Project Manager
Developer Advocate
DAO Advocate

OpenZeppelin

Thank you!

Learn more

openzeppelin.com/**contracts**

openzeppelin.com/**defender**

forum.openzeppelin.com

docs.openzeppelin.com

Contact

@OpenZeppelin

contact@openzeppelin.com