

Community Call #1

Francisco Giordano

frangio@openzeppelin.com



Hadrien Croubois

hadrien@openzeppelin.com



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.































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.



Why do a community call

Give visibility to new features & improvements

Involve you in the process

Provide feedback

Request features

Engage discussion

Review the code



OpenZeppelin Contracts Security Process

Internal code reviews

99% test coverage

External audit and formal verification

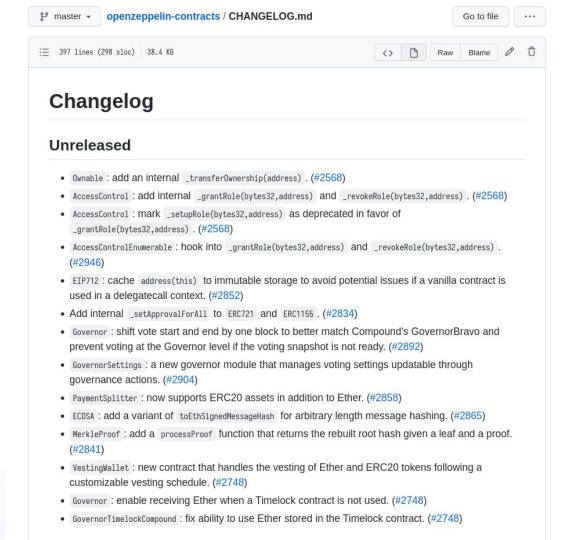
Community review

- → Release candidates
- → Open Review Period









Add internal functions to Ownable, AccessControl, ERC721 and ERC1155

Goal: Allow the developers to trigger "internal operations", without access restriction, in certain cases:

- **Ownable:** TransferOwnership "withSignature".
- AccessControl: GrantRole / RevokeRole "withSignature".
- **ERC721/ERC1155:** SetApprovalForAll "WithSignature".
- **Other:** SellableOwnable (transfer ownership on payment).

```
bstract contract OwnableWithSig is Ownable, EIP712 {
  using Counters for Counters.Counter;
  mapping(address => Counters.Counter) private nonces;
  // solhint-disable-next-line var-name-mixedcase
  bytes32 private immutable TRANSFEROWNERSHIP TYPEHASH =
      keccak256("TransferOwnership(address newOwner.uint256 nonce)"):
      address newOwner,
      uint8 v.
      bytes32 s
  ) public virtual {
          owner() == ECDSA.recover(
               hashTypedDataV4(keccak256(abi.encode(
                   TRANSFEROWNERSHIP TYPEHASH,
                  newOwner.
                   useNonce(owner())
          "Invalid signature"
       transferOwnership(newOwner);
   * +++ Copied from token/ERC20/extensions/ERC20Permit.sol +++
  function nonces(address owner) public view virtual override returns (uint256) {
      return nonces[owner].current();
  // solhint-disable-next-line func-name-mixedcase
  function DOMAIN SEPARATOR() external view override returns (bytes32) {
      return domainSeparatorV4();
  function useNonce(address owner) internal virtual returns (uint256 current) {
      Counters.Counter storage nonce = nonces[owner];
      current = nonce.current():
      nonce.increment():
```

GovernorSettings

Goal: Make governance parameters settable via governance proposals.

```
function setVotingDelay(uint256 newVotingDelay) public onlyGovernance;
function setVotingPeriod(uint256 newVotingPeriod) public onlyGovernance;
function setProposalThreshold(uint256 newProposalThreshold) public onlyGovernance;
modifier onlyGovernance() {
    require(_msgSender() == _executor());
    _;
}
```



ETH funds in Governor & Compound Timelock

Goal: Allow a Governor to handle ETH funds when using the Compound Timelock.

Issue identified by <u>Tally</u>.

```
uint256 eta = proposalEta(proposalId);
    require(eta > 0, "GovernorTimelockCompound: proposal not yet qu
+ Address.sendValue(payable(_timelock), msg.value);
    for (uint256 i = 0; i < targets.length; ++i) {
        _timelock.executeTransaction{value: values[i]}(targets[i],
        _timelock.executeTransaction(targets[i], values[i], "", cal
}</pre>
```

Governor vote start and end shifted

Goal: Correct small difference with GovernorBravo, and mitigate a risk with bad vote tokens.

Reported by @zgfzgf.

EIP712: cache address(this)

Goal: Add address check, on top of chain id check before reading from cache to prevent potential issues if contract is used through proxy

Advisory

When using OpenZeppelin Contracts through proxies, use:

@openzeppelin/contracts-upgradeable

PR proposed by <a>@k06a

ECDSA.toEthSignedMessageHash(bytes)

Goal: Expand support of EIP191 signed messages to arbitrary length buffers.

PR proposed by @td-bn

```
/**

* @dev Returns an Ethereum Signed Message, created from a 'hash'. This

* produces hash corresponding to the one signed with the

* https://eth.wiki/json-rpc/API#eth_sign['eth_sign']

* JSON-RPC method as part of EIP-191.

* See {recover}.

*/

function toEthSignedMessageHash(bytes32 hash) internal pure returns (bytes32) {

// 32 is the length in bytes of hash,

// enforced by the type signature above
return keccak256(abi.encodePacked("\x19Ethereum Signed Message:\n32", hash));

}

/**

* @dev Returns an Ethereum Signed Message, created from 's'. This

* produces hash corresponding to the one signed with the

* https://eth.wiki/json-rpc/API#eth_sign['eth_sign']

* JSON-RPC method as part of EIP-191.

* See {recover}.

*/

function toEthSignedMessageHash(bytes memory s) internal pure returns (bytes32) {

return keccak256(abi.encodePacked("\x19Ethereum Signed Message:\n", Strings.toString(s.length), s));

}
```

MerkleProof.processProof(bytes32[], bytes32)

Goal: Improve support for Merkle trees, particularly when multiple trees are valid

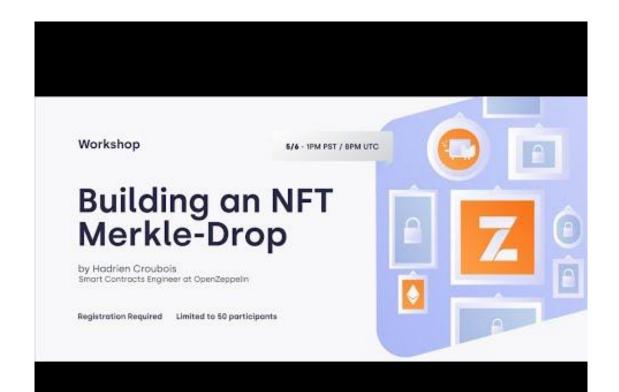
```
library MerkleProof {
    * defined by 'root'. For this, a 'proof' must be provided, containing
    * sibling hashes on the branch from the leaf to the root of the tree. Each
     * pair of leaves and each pair of pre-images are assumed to be sorted.
    function verify(
       bytes32[] memory proof,
       bytes32 root,
       bytes32 leaf
   ) internal pure returns (bool) {
       return processProof(proof, leaf) == root;
    * from `leaf` using `proof`. A `proof` is valid if and only if the rebuilt
    * hash matches the root of the tree. When processing the proof, the pairs
     * of leafs & pre-images are assumed to be sorted.
    function processProof(bytes32[] memory proof, bytes32 leaf) internal pure returns (bytes32) {
       bytes32 computedHash = leaf;
       for (uint256 i = 0; i < proof.length; i++) {</pre>
           bytes32 proofElement = proof[i];
           if (computedHash <= proofElement) {</pre>
               computedHash = keccak256(abi.encodePacked(computedHash, proofElement));
           } else {
               computedHash = keccak256(abi.encodePacked(proofElement, computedHash));
        return computedHash;
```

MerkleProof.processProof(bytes32[], bytes32)

Goal: Improve support for Merkle trees, particularly when multiple trees are valid

function **verify**(bytes32[] memory **proof**, bytes32 **root**, bytes32 **leaf**) internal pure returns (**bool**); function **processProof**(bytes32[] memory **proof**, bytes32 **leaf**) internal pure returns (**bytes32**);

```
function release(Schedule memory schedule, bytes32[] memory proof) public {
 // check input validity
 bytes32 leaf = keccak256(abi.encode(schedule));
 bytes32 drop = MerkleProof.processProof(proof, leaf);
 require(_airdrops[drop]);
 // compute vesting remains
 uint256 releasable = vestedAmount(schedule, uint64(block.timestamp)) - _released[leaf];
 _released[leaf] += releasable;
 // emit notification
 emit TokensReleased(drop, schedule.token, schedule.recipient, schedule.amount);
 // do release (might have reentrancy)
 SafeERC20.safeTransfer(schedule.token, schedule.recipient, releasable);
function vestedAmount(Schedule memory schedule, uint64 timestamp) public pure returns (uint256) {
 return timestamp < schedule.start + schedule.cliff
 ? 0
 : Math.min(
     schedule.amount,
     schedule.amount * (timestamp - schedule.start) / schedule.duration
```



https://www.youtube.com/watch?v=SF-XOwWIwRo

Payment Splitter support for ERC20 tokens

Goal: Add support for ERC20 tokens in the payment splitter.

- Storage layout is compatible, making upgrades possible.
- Payee management is unmodified.
- Follow the "pull payment" model.

Note: This contract assumes that ERC20 tokens will behave similarly to native tokens (Ether). Rebasing tokens, and tokens that apply fees during transfers, are likely to not be supported as expected. If in doubt, we encourage you to run tests before sending real value to this contract.

```
require(_shares[account] > 0, "PaymentSplitter: account has no shares");
   uint256 totalReceived = address(this).balance + totalReleased();
   uint256 payment = _pendingPayment(account, totalReceived, released(account));
    require(payment != 0, "PaymentSplitter: account is not due payment");
   Address.sendValue(account, payment);
    emit PaymentReleased(account, payment);
 * percentage of the total shares and their previous withdrawals. `token` must be the address of an IERC20
function release(IERC20 token, address account) public virtual {
   require( shares[account] > 0, "PaymentSplitter: account has no shares");
   uint256 totalReceived = token.balanceOf(address(this)) + totalReleased(token);
   uint256 payment = _pendingPayment(account, totalReceived, released(token, account));
    require(payment != 0, "PaymentSplitter: account is not due payment");
    _erc20Released[token][account] += payment;
    erc20TotalReleased[token] += payment;
    emit ERC20PaymentReleased(token, account, payment);
function _pendingPayment(
   uint256 alreadyReleased
) private view returns (uint256) {
    return (totalReceived * _shares[account]) / _totalShares - alreadyReleased;
```

Vesting Wallet

```
Standalone contract.
Supported assets (Ether/ERC20) are released following a customizable vesting schedule.
constructor(address beneficiary, uint64 startTimestamp, uint64 durationSeconds)
release() public;
release(address token) public;
vestedAmount(uint64 timestamp) public view returns (uint256);
vestedAmount(address token, uint64 timestamp) public view returns (uint256);
_vestingSchedule(uint256 totalAllocation, uint64 timestamp) internal view virtual returns
(uint256);
```

OpenZeppelin

Contribute

Formal bug bounty program

More details to come very soon

Open Review Period

4.4 Release Candidate - Available Now!

https://zpl.in/contracts/4.4

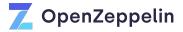


docs.openzeppelin.com blog.openzeppelin.com forum.openzeppelin.com defender.openzeppelin.com

We're hiring!

zpl.in/**join**

Blockchain Security Engineer
Full Stack Ethereum Developer
Site Reliability Engineer
Open Source Developer
Application UX Designer
Technical Project Manager



Thank you!





Learn more

openzeppelin.com/contracts forum.openzeppelin.com docs.openzeppelin.com

Contact

