Workshop #2

Secure Development by OpenZeppelin

08/05 - 12PM PST / 7PM UTC

Strategies for Secure Access Controls

Martin Abbatemarco

Security Researcher at OpenZeppelin

REGISTRATION REQUIRED LIMITED TO 50 ATENDEES

Series of sessions

Secure Development

The dangers of token integration ✓

Strategies for secure access controls

The dangers of price oracles

and more!

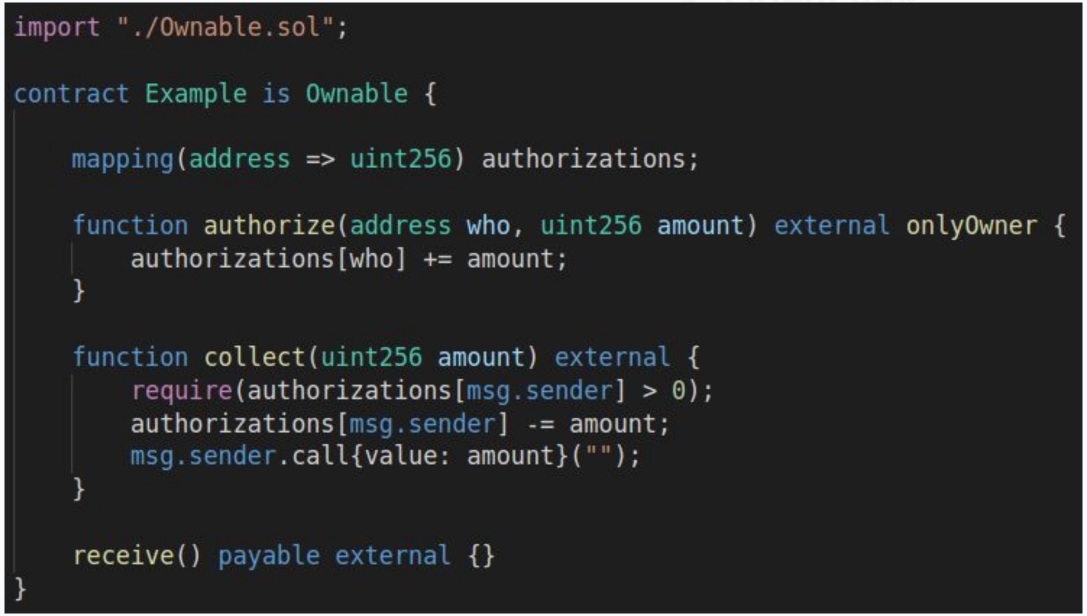
$$\begin{tikzcd}
\text{centralized} \arrow[r] & \text{decentralized} \\
& \text{Progressive decentralization ?} \arrow[from=1-1, to=2-2, bend right]
\end{tikzcd}$$

consistent and reliable

access controls

Problem(s) ?

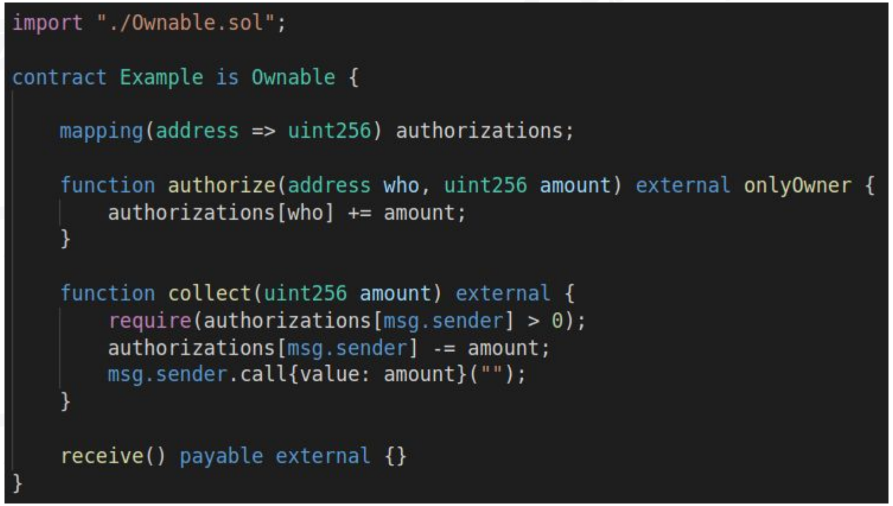
(60 seconds)



```
import "./Ownable.sol";

contract Example is Ownable {

    mapping(address => uint256) authorizations;

    function authorize(address who, uint256 amount) external onlyOwner {
        authorizations[who] += amount;
    }

    function collect(uint256 amount) external {
        require(authorizations[msg.sender] > 0);
        authorizations[msg.sender] -= amount;
        msg.sender.call{value: amount}("");
    }

    receive() payable external {}
}
```

question the ~~law~~ code

some initial triggers



```
import "./Ownable.sol";

contract Example is Ownable {

    mapping(address => uint256) authorizations;

    function authorize(address who, uint256 amount) external onlyOwner {
        authorizations[who] += amount;
    }

    function collect(uint256 amount) external {
        require(authorizations[msg.sender] > 0);
        authorizations[msg.sender] -= amount;
        msg.sender.call{value: amount}("");
    }

    receive() payable external {}
}
```

Does it even compile ?

What Solidity version ? Potential overflows ?

No docstrings - what's this *supposed* to do ?

Wait, what does the "Ownable" contract look like ?

No visibility on state variables ?

No events ?

No error messages ?

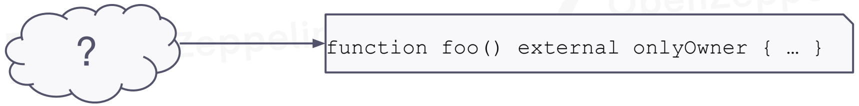
Not checking return value in low-level call ?

Owner authorizing itself ? Rug-pulling scenarios ?

Who controls the privileged account ?

...

more questions



```
function foo() external onlyOwner { … }
```

Single private key ?

Multiple private keys through a multisig ?

Governance ? Fully open or limited ?

Other contract of the system ?

Do they hold other roles in the system ?

Is there a timelock mechanism behind it ? Can anyone bypass it ?

How are those keys stored ? Backups ?

Who's got access to them ?

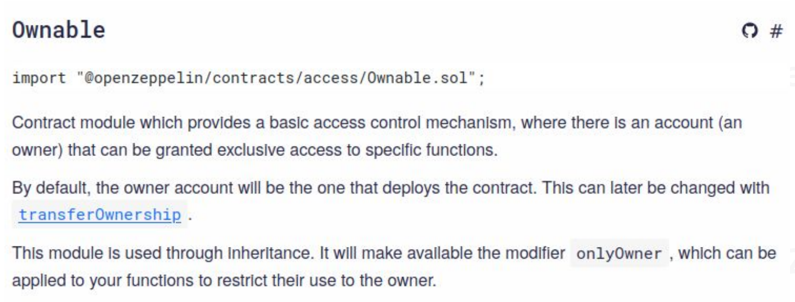
Are their actions logged and monitored ?

Where / how are keys generated ?

Standard signing or custom implementation ?

ownership

ownership



Ownable

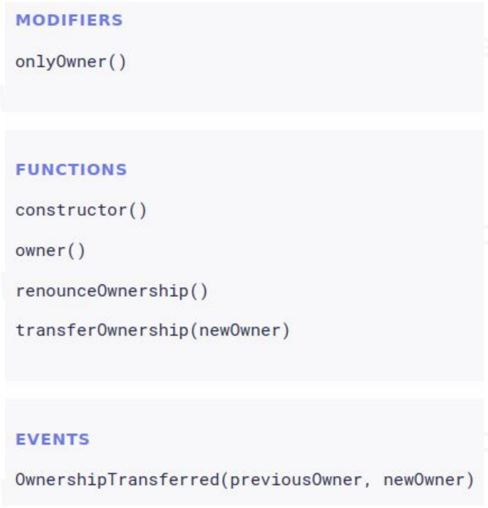
```
import "@openzeppelin/contracts/access/Ownable.sol";
```

Contract module which provides a basic access control mechanism, where there is an account (an owner) that can be granted exclusive access to specific functions.

By default, the owner account will be the one that deploys the contract. This can later be changed with `transferOwnership` .

This module is used through inheritance. It will make available the modifier `onlyOwner` , which can be applied to your functions to restrict their use to the owner.

https://docs.openzeppelin.com/contracts/4.x/api/access#Ownable



MODIFIERS

`onlyOwner()`

FUNCTIONS

`constructor()`

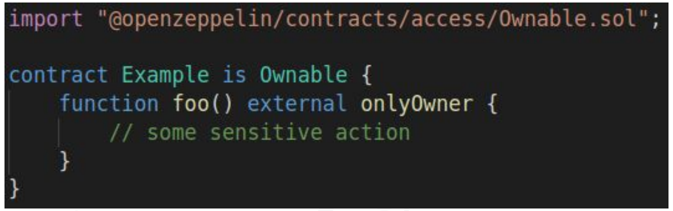
`owner()`

`renounceOwnership()`

`transferOwnership(newOwner)`

EVENTS

`OwnershipTransferred(previousOwner, newOwner)`



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

contract Example is Ownable {
    function foo() external onlyOwner {
        // some sensitive action
    }
}
```

ownership

```
contract DSAuth is DSAuthEvents {
    DSAuthority  public  authority;
    address      public  owner;

    constructor() public {
        owner = msg.sender;
        emit LogSetOwner(msg.sender);
    }

    function setOwner(address owner_)
        public
        auth
    {
        owner = owner_;
        emit LogSetOwner(owner);
    }


    modifier auth {
        require(isAuthorized(msg.sender, msg.sig), "ds-auth-unauthorized");
        _;
    }
```

```
function setAuthority(DSAuthority authority_)
    public
    auth
{
    authority = authority_;
    emit LogSetAuthority(address(authority));
}
```

Allows setting an “authority” contract for finer-grained auth controls

https://github.com/dapphub/ds-auth/blob/master/src/auth.sol

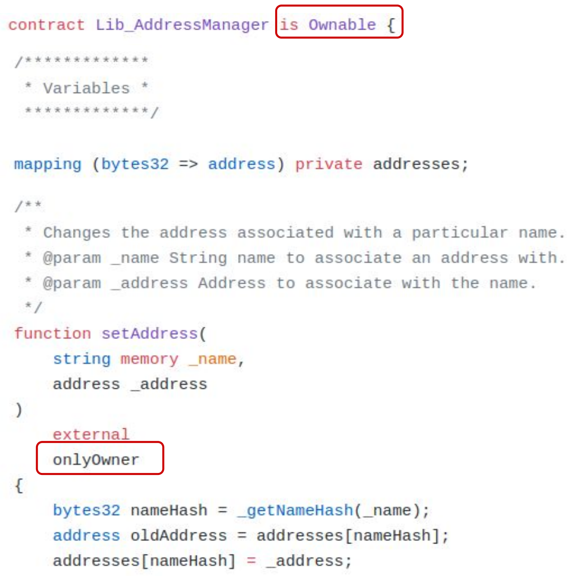
It’s fine to use onlyOwner

Document its powers, and be transparent with your community

ownership

ownership

It’s fine to use onlyOwner

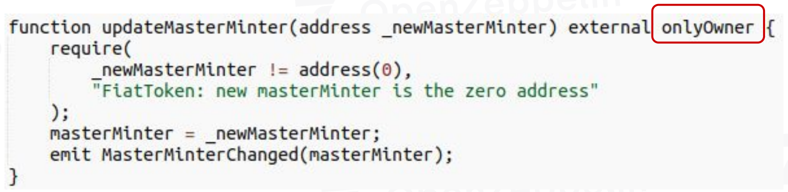


```
contract Lib_AddressManager is Ownable {

 /*************
  * Variables *
  *************/

 mapping (bytes32 => address) private addresses;

 /**
  * Changes the address associated with a particular name.
  * @param _name String name to associate an address with.
  * @param _address Address to associate with the name.
  */
 function setAddress(
     string memory _name,
     address _address
 )
     external
     onlyOwner
 {
     bytes32 nameHash = _getNameHash(_name);
     address oldAddress = addresses[nameHash];
     addresses[nameHash] = _address;
```

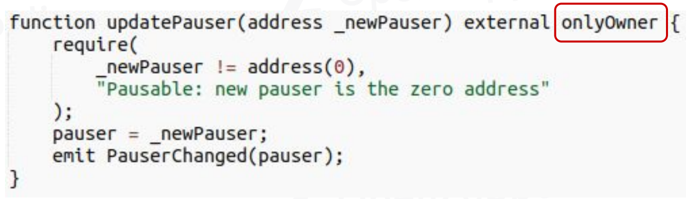
AddressManager contract in Optimistic Ethereum

It’s fine to use onlyOwner

ownership



```
function updateMasterMinter(address _newMasterMinter) external onlyOwner {
    require(
        _newMasterMinter != address(0),
        "FiatToken: new masterMinter is the zero address"
    );
    masterMinter = _newMasterMinter;
    emit MasterMinterChanged(masterMinter);
}
```

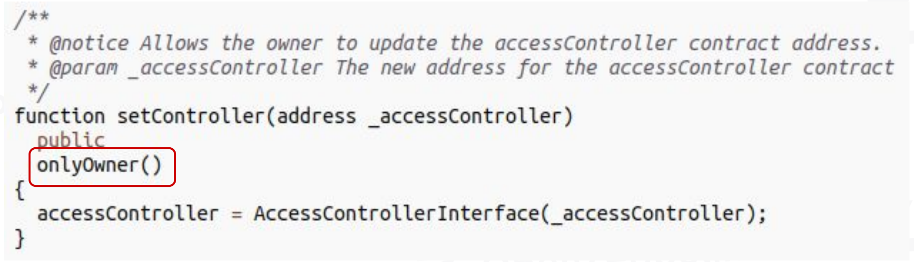


```
function updatePauser(address _newPauser) external onlyOwner {
    require(
        _newPauser != address(0),
        "Pausable: new pauser is the zero address"
    );
    pauser = _newPauser;
    emit PauserChanged(pauser);
}
```

USDC

It’s fine to use onlyOwner

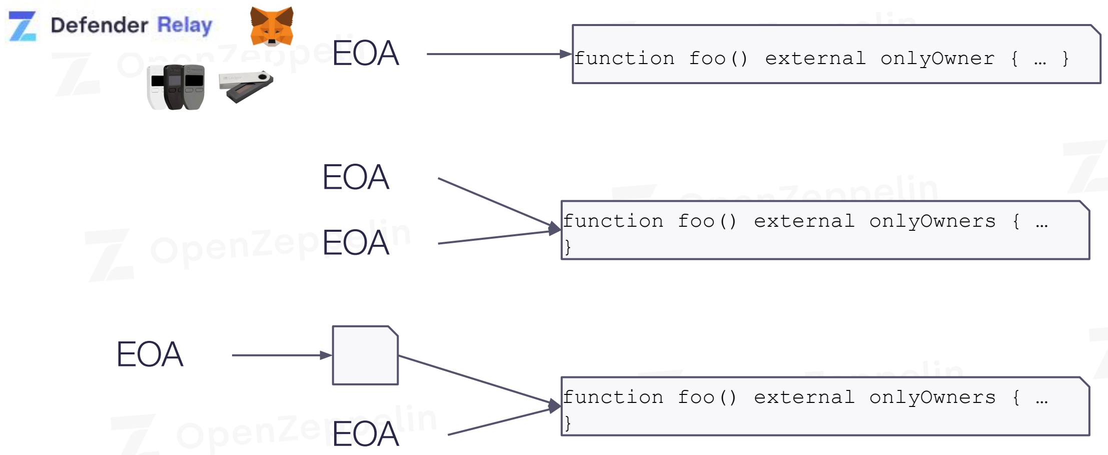
ownership



```
/**
 * @notice Allows the owner to update the accessController contract address.
 * @param _accessController The new address for the accessController contract
 */
function setController(address _accessController)
  public
  onlyOwner()
{
  accessController = AccessControllerInterface(_accessController);
}
```

[ChainLink ETH/USD feed]

ownership

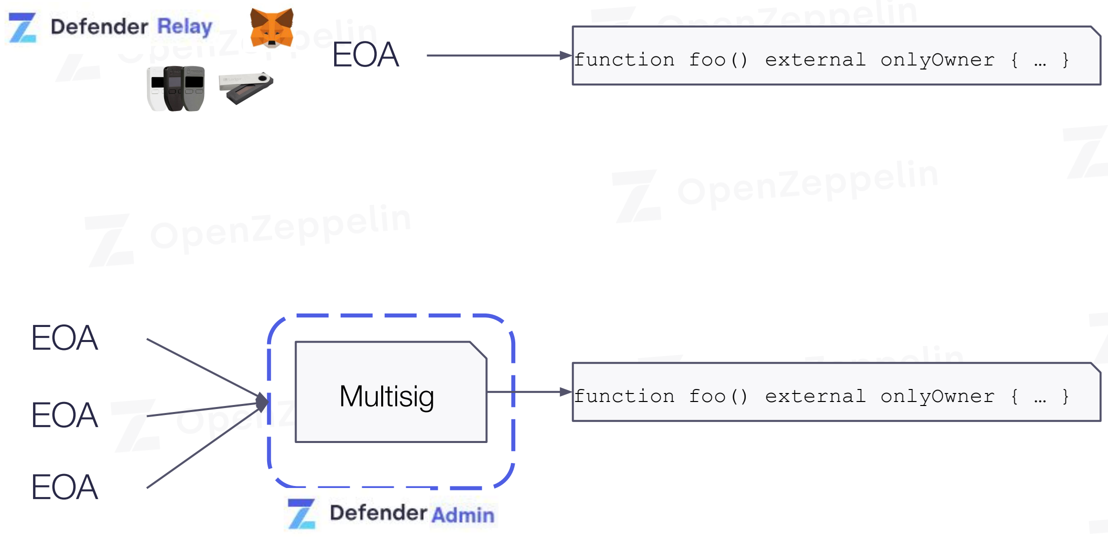


```
function foo() external onlyOwner { … }
```

```
function foo() external onlyOwners { …
}
```

```
function foo() external onlyOwners { …
}
```

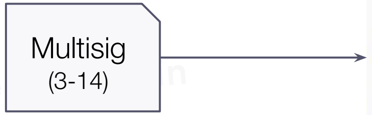
ownership


Defender Relay
EOA
function foo() external onlyOwner { … }
EOA
EOA
EOA
Multisig
function foo() external onlyOwner { … }
Defender Admin

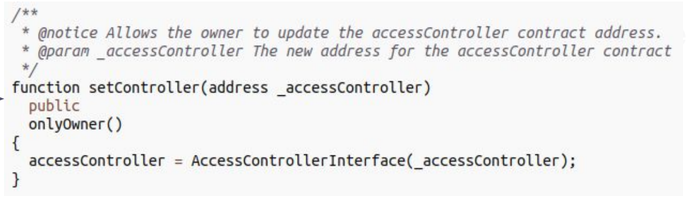
[...]

ownership

It’s fine to use onlyOwner



0x21f73D42Eb58Ba49dDB685dc29D3bF5c0f0373CA



```
/**
 * @notice Allows the owner to update the accessController contract address.
 * @param _accessController The new address for the accessController contract
 */
function setController(address _accessController)
  public
  onlyOwner()
{
  accessController = AccessControllerInterface(_accessController);
}
```

[ChainLink ETH/USD feed]

[ChainLink BTC/USD feed]

[ChainLink AAVE/USD feed]

...

^ tiny spoiler for next session about oracles 😏

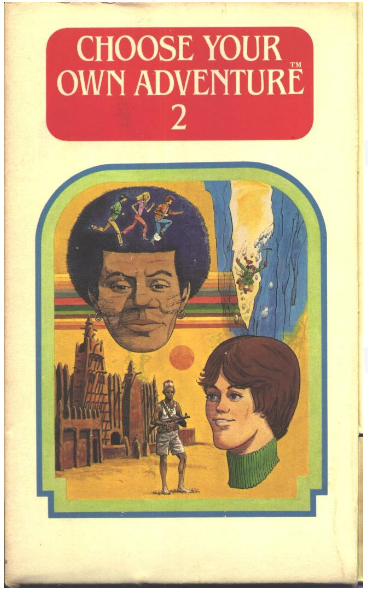
roles

roles

different sensitive actions and levels of authorization

roles

A food-named mintable token
to be pumped, dumped and forked in 2 hours after launch
that can be paused and upgraded

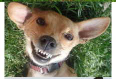


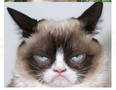
can do everything

→ mint and pause

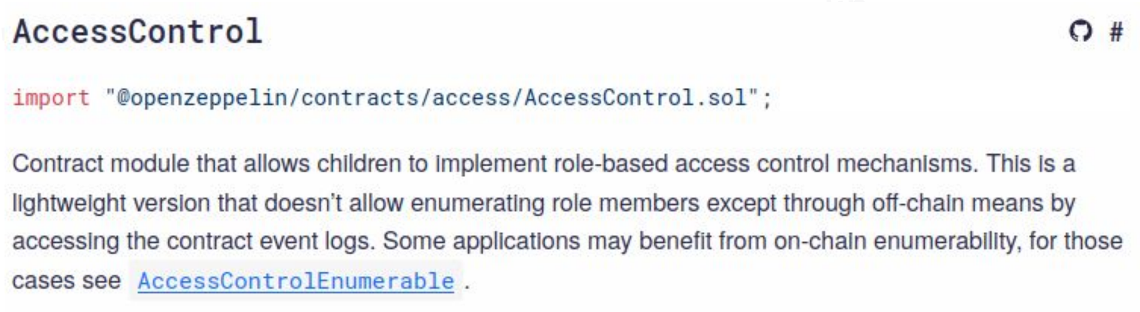
→ upgrade

→ mint

 → pause

 → upgrade

roles

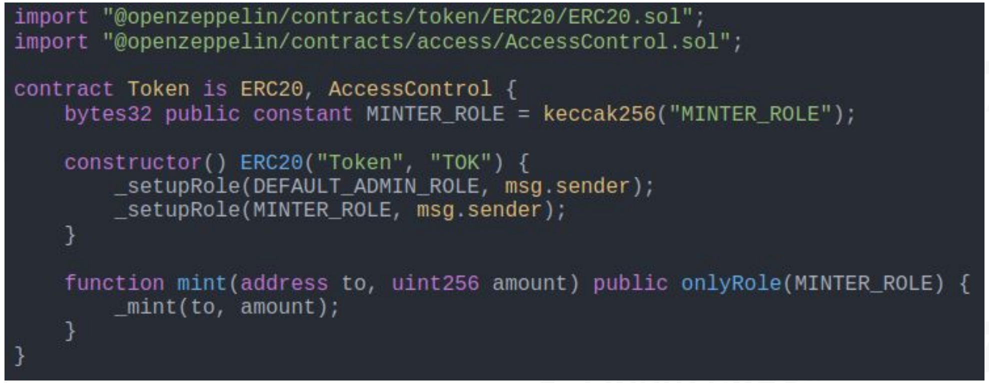


AccessControl

```
import "@openzeppelin/contracts/access/AccessControl.sol";
```

Contract module that allows children to implement role-based access control mechanisms. This is a lightweight version that doesn't allow enumerating role members except through off-chain means by accessing the contract event logs. Some applications may benefit from on-chain enumerability, for those cases see AccessControlEnumerable .

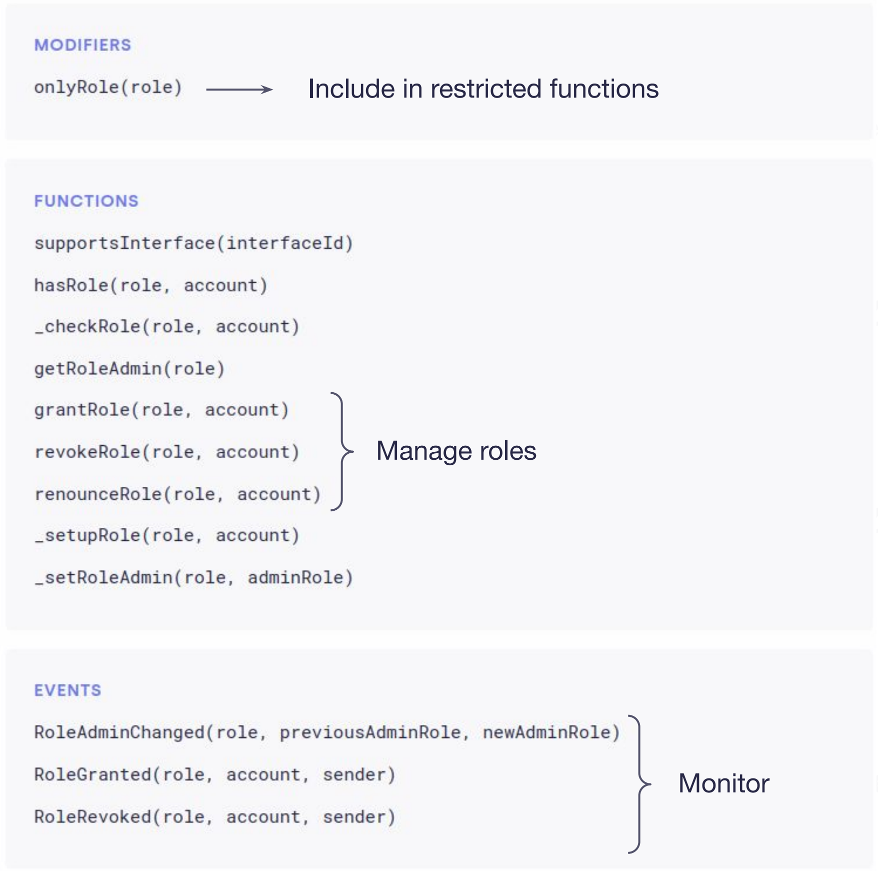
https://docs.openzeppelin.com/contracts/4.x/api/access#AccessControl



```
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/access/AccessControl.sol";

contract Token is ERC20, AccessControl {
    bytes32 public constant MINTER_ROLE = keccak256("MINTER_ROLE");

    constructor() ERC20("Token", "TOK") {
        _setupRole(DEFAULT_ADMIN_ROLE, msg.sender);
        _setupRole(MINTER_ROLE, msg.sender);
    }

    function mint(address to, uint256 amount) public onlyRole(MINTER_ROLE) {
        _mint(to, amount);
    }
}
```

roles

Some out-of-the-box features of AccessControl



MODIFIERS

`onlyRole(role)` → Include in restricted functions

FUNCTIONS

```
supportsInterface(interfaceId)
hasRole(role, account)
_checkRole(role, account)
getRoleAdmin(role)
grantRole(role, account)
revokeRole(role, account)
renounceRole(role, account)
_setupRole(role, account)
_setRoleAdmin(role, adminRole)
```

$\left.\begin{array}{l}\texttt{grantRole(role, account)}\\ \texttt{revokeRole(role, account)}\\ \texttt{renounceRole(role, account)}\end{array}\right\}$ Manage roles

EVENTS

```
RoleAdminChanged(role, previousAdminRole, newAdminRole)
RoleGranted(role, account, sender)
RoleRevoked(role, account, sender)
```

$\left.\begin{array}{l}\texttt{RoleAdminChanged}\\ \texttt{RoleGranted}\\ \texttt{RoleRevoked}\end{array}\right\}$ Monitor

https://docs.openzeppelin.com/contracts/4.x/api/access#AccessControl

roles

How long does it take to code a **secure** upgradeable mintable ERC721 **with role-based access controls** ?

Less than a day

1 to 5 days

More than 5 days

roles

zpl.in/wizard

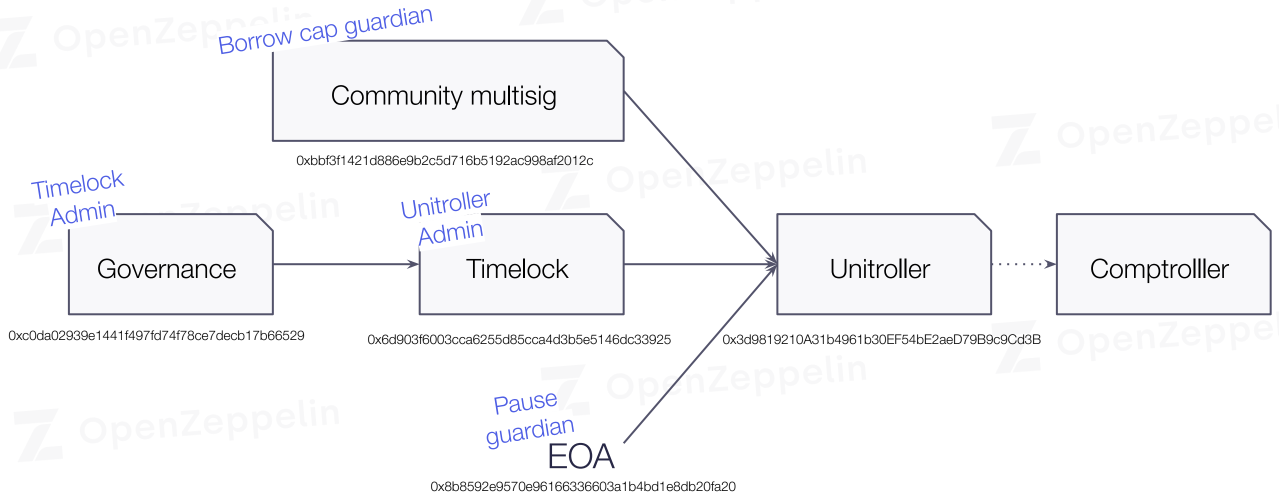
It’s fine to use roles

Document their powers, and be transparent with your community

roles

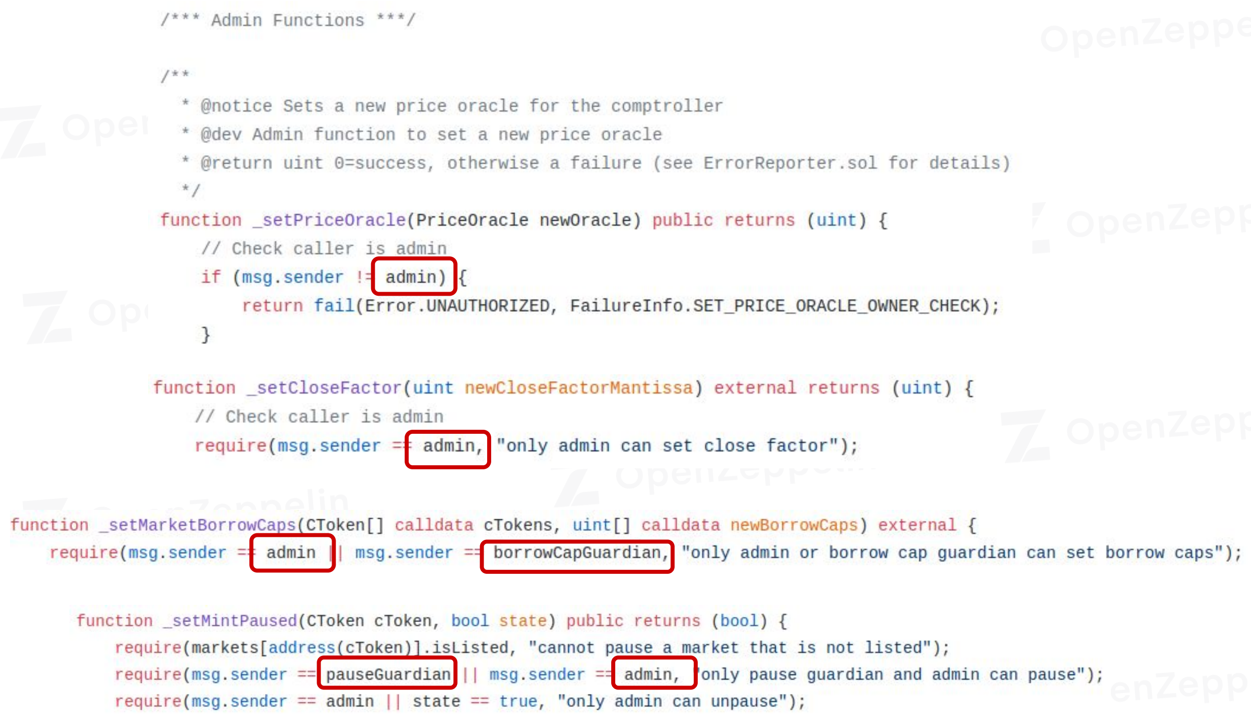
roles

It’s fine to use roles


Borrow cap guardian
Community multisig
0xbbf3f1421d886e9b2c5d716b5192ac998af2012c
Timelock Admin
Governance
0xc0da02939e1441f497fd74f78ce7decb17b66529
Unitroller Admin
Timelock
0x6d903f6003cca6255d85cca4d3b5e5146dc33925
Unitroller
0x3d9819210A31b4961b30EF54bE2aeD79B9c9Cd3B
Comptrolller
Pause guardian
EOA
0x8b8592e9570e96166336603a1b4bd1e8db20fa20

roles

It’s fine to use roles



```
/*** Admin Functions ***/

/**
  * @notice Sets a new price oracle for the comptroller
  * @dev Admin function to set a new price oracle
  * @return uint 0=success, otherwise a failure (see ErrorReporter.sol for details)
  */
function _setPriceOracle(PriceOracle newOracle) public returns (uint) {
    // Check caller is admin
    if (msg.sender != admin) {
        return fail(Error.UNAUTHORIZED, FailureInfo.SET_PRICE_ORACLE_OWNER_CHECK);
    }

function _setCloseFactor(uint newCloseFactorMantissa) external returns (uint) {
    // Check caller is admin
    require(msg.sender == admin, "only admin can set close factor");

function _setMarketBorrowCaps(CToken[] calldata cTokens, uint[] calldata newBorrowCaps) external {
    require(msg.sender == admin || msg.sender == borrowCapGuardian, "only admin or borrow cap guardian can set borrow caps");

function _setMintPaused(CToken cToken, bool state) public returns (bool) {
    require(markets[address(cToken)].isListed, "cannot pause a market that is not listed");
    require(msg.sender == pauseGuardian || msg.sender == admin, "only pause guardian and admin can pause");
    require(msg.sender == admin || state == true, "only admin can unpause");
```

Compound protocol’s Comptroller

roles

too many roles → difficult to manage & coordinate actions

too few roles → dangerously powerful accounts

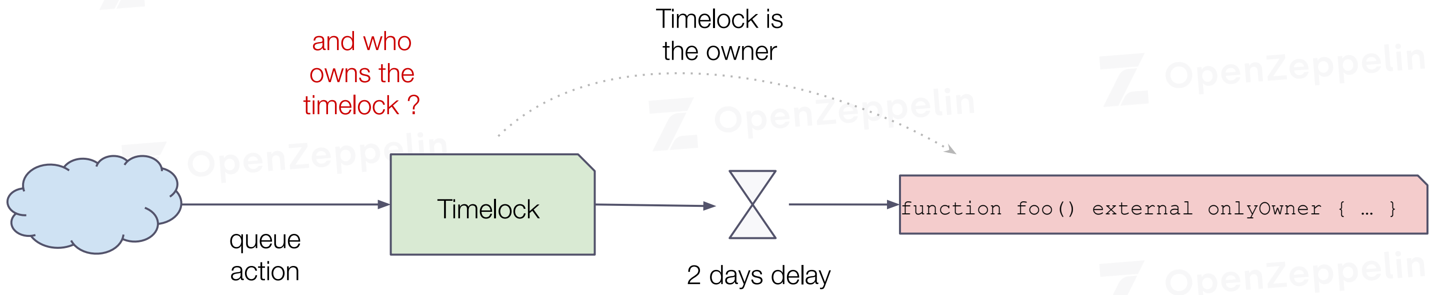
timelocks

timelocks

allows time-delayed opt-out changes

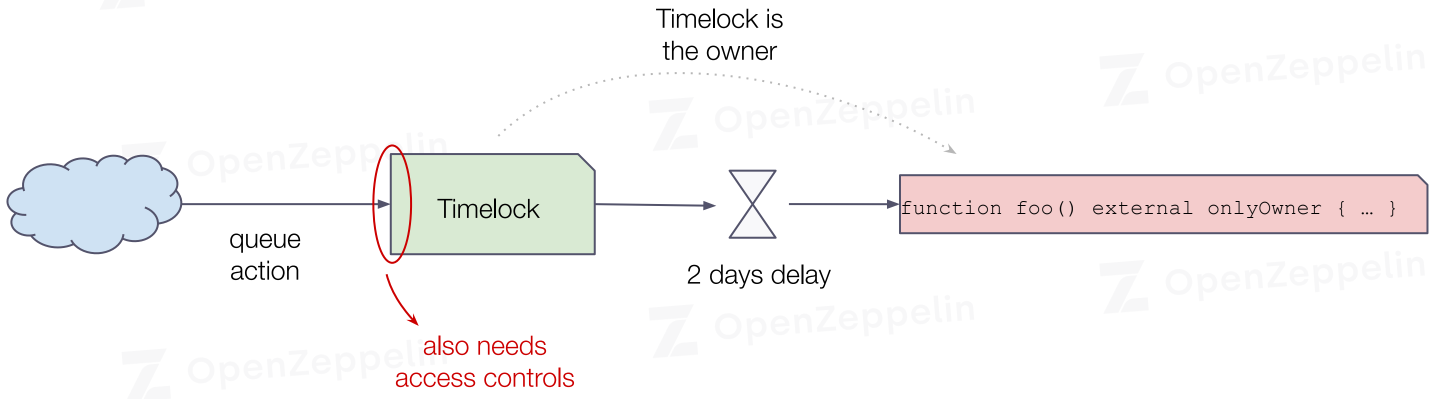
giving users time to react

timelocks



```
function foo() external onlyOwner { … }
```

timelocks



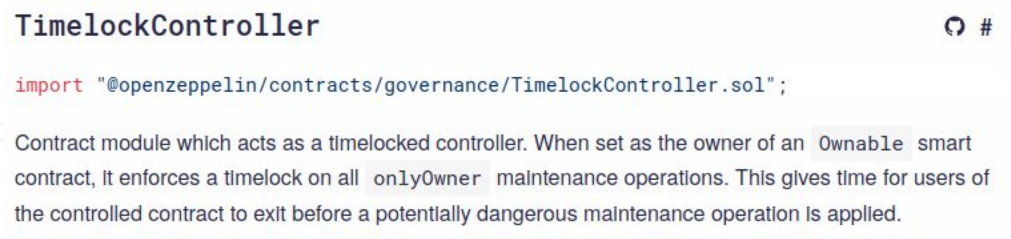
```
function foo() external onlyOwner { … }
```

timelocks

how long does it take to code a Timelock contract?

actually, no need to code it

timelocks

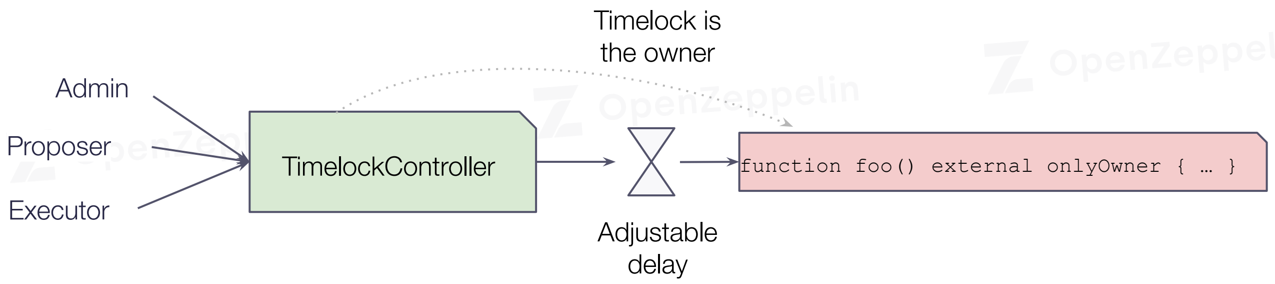


TimelockController

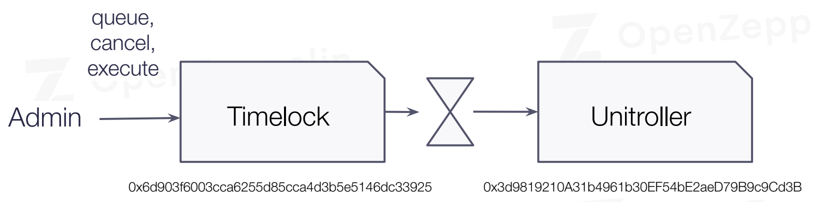
```
import "@openzeppelin/contracts/governance/TimelockController.sol";
```

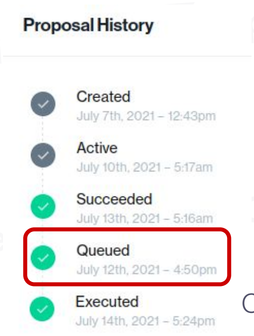
Contract module which acts as a timelocked controller. When set as the owner of an `Ownable` smart contract, it enforces a timelock on all `onlyOwner` maintenance operations. This gives time for users of the controlled contract to exit before a potentially dangerous maintenance operation is applied.

https://docs.openzeppelin.com/contracts/4.x/api/governance#TimelockController



It’s fine to use timelocks


queue,
cancel,
execute
Admin
Timelock
Unitroller
0x6d903f6003cca6255d85cca4d3b5e5146dc33925
0x3d9819210A31b4961b30EF54bE2aeD79B9c9Cd3B



Proposal History

- Created — July 7th, 2021 – 12:43pm
- Active — July 10th, 2021 – 5:17am
- Succeeded — July 13th, 2021 – 5:16am
- Queued — July 12th, 2021 – 4:50pm
- Executed — July 14th, 2021 – 5:24pm

proposal queued in the Timelock by Compound’s governance

timelocks

```
function acceptAdmin() public {
    require(msg.sender == pendingAdmin, "Timelock::acceptAdmin: Call must come from pendingAdmin.");
    admin = msg.sender;
    pendingAdmin = address(0);

    emit NewAdmin(admin);
}

function setPendingAdmin(address pendingAdmin_) public {
    require(msg.sender == address(this), "Timelock::setPendingAdmin: Call must come from Timelock.");
    pendingAdmin = pendingAdmin_;

    emit NewPendingAdmin(pendingAdmin);
}
```

offer-accept pattern for admin

changes to the timelock are timelocked by the timelock itself !

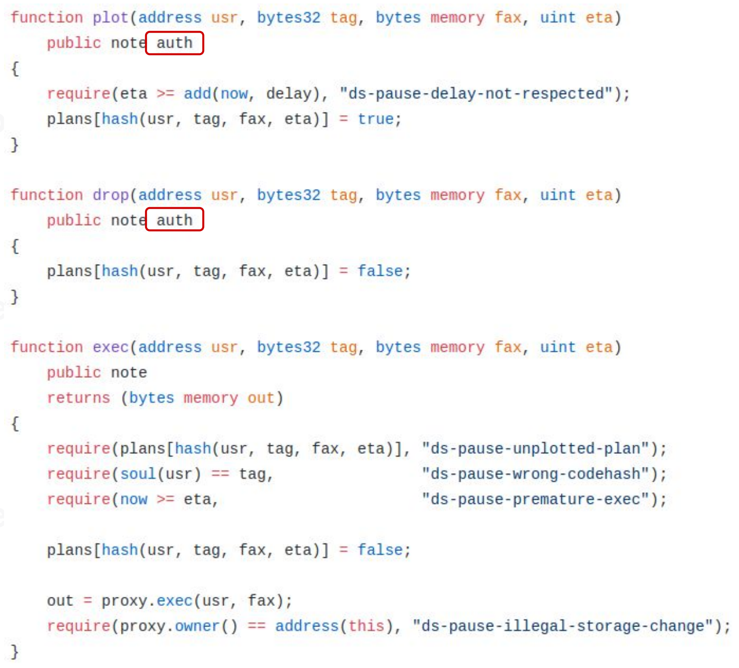
timelocks

It’s fine to use timelocks

queue

cancel

execute



```
function plot(address usr, bytes32 tag, bytes memory fax, uint eta)
    public note auth
{
    require(eta >= add(now, delay), "ds-pause-delay-not-respected");
    plans[hash(usr, tag, fax, eta)] = true;
}

function drop(address usr, bytes32 tag, bytes memory fax, uint eta)
    public note auth
{
    plans[hash(usr, tag, fax, eta)] = false;
}

function exec(address usr, bytes32 tag, bytes memory fax, uint eta)
    public note
    returns (bytes memory out)
{
    require(plans[hash(usr, tag, fax, eta)], "ds-pause-unplotted-plan");
    require(soul(usr) == tag,                "ds-pause-wrong-codehash");
    require(now >= eta,                      "ds-pause-premature-exec");

    plans[hash(usr, tag, fax, eta)] = false;

    out = proxy.exec(usr, fax);
    require(proxy.owner() == address(this), "ds-pause-illegal-storage-change");
}
```

https://github.com/dapphub/ds-pause/blob/master/src/pause.sol

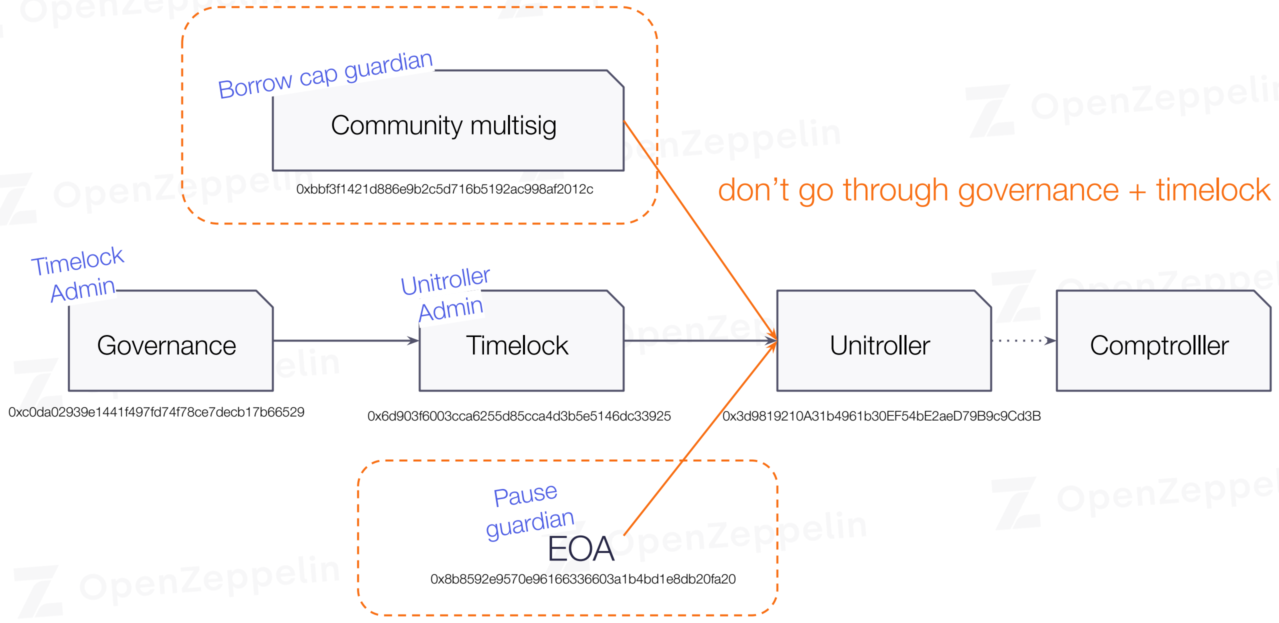
https://docs.makerdao.com/smart-contract-modules/governance-module/pause-detailed-documentation

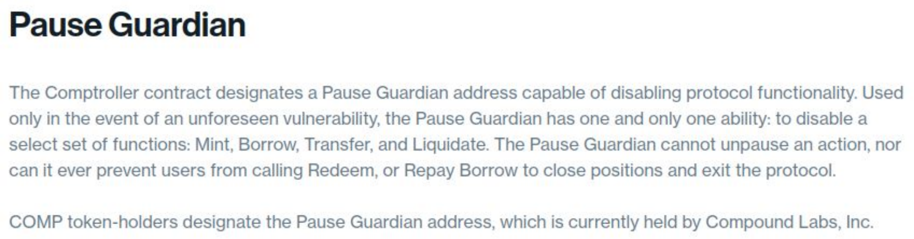
timelocks

It’s fine to use timelocks,

but you might need to bypass them

timelocks


Borrow cap guardian
Community multisig
0xbbf3f1421d886e9b2c5d716b5192ac998af2012c
don't go through governance + timelock
Timelock Admin
Governance
0xc0da02939e1441f497fd74f78ce7decb17b66529
Unitroller Admin
Timelock
0x6d903f6003cca6255d85cca4d3b5e5146dc33925
Unitroller
0x3d9819210A31b4961b30EF54bE2aeD79B9c9Cd3B
Comptrolller
Pause guardian
EOA
0x8b8592e9570e96166336603a1b4bd1e8db20fa20



Pause Guardian

The Comptroller contract designates a Pause Guardian address capable of disabling protocol functionality. Used only in the event of an unforeseen vulnerability, the Pause Guardian has one and only one ability: to disable a select set of functions: Mint, Borrow, Transfer, and Liquidate. The Pause Guardian cannot unpause an action, nor can it ever prevent users from calling Redeem, or Repay Borrow to close positions and exit the protocol.

COMP token-holders designate the Pause Guardian address, which is currently held by Compound Labs, Inc.

https://compound.finance/docs/governance#pause-guardian

governance

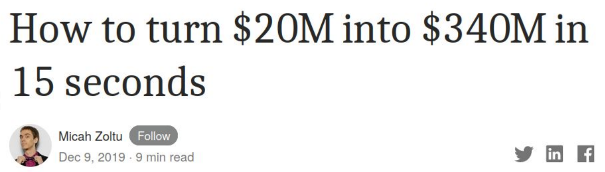
governance

complexity tends to be

on mechanism and incentive design

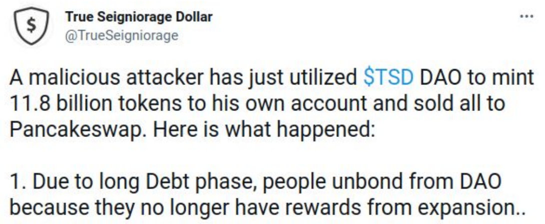
rather than on actual implementation

governance


How to turn $20M into $340M in 15 seconds

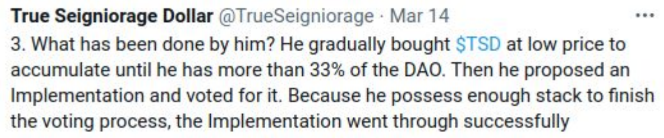
Micah Zoltu Follow
Dec 9, 2019 · 9 min read

https://medium.com/coinmonks/how-to-turn-20m-into-340m-in-15-seconds-48d161a42311


True Seigniorage Dollar
@TrueSeigniorage

A malicious attacker has just utilized $TSD DAO to mint 11.8 billion tokens to his own account and sold all to Pancakeswap. Here is what happened:

1. Due to long Debt phase, people unbond from DAO because they no longer have rewards from expansion..


True Seigniorage Dollar @TrueSeigniorage · Mar 14

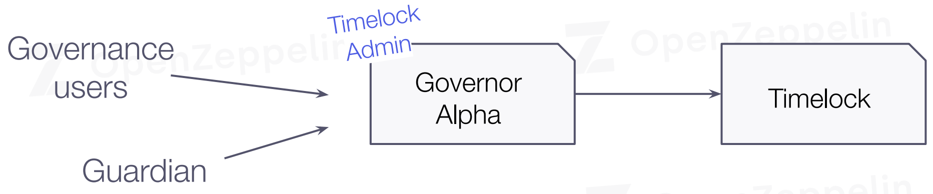
3. What has been done by him? He gradually bought $TSD at low price to accumulate until he has more than 33% of the DAO. Then he proposed an Implementation and voted for it. Because he possess enough stack to finish the voting process, the Implementation went through successfully

https://twitter.com/trueseigniorage/status/1370956726489415683

governance

- A contract that is allowed to execute sensitive on others
- Propose actions and vote on them
- Action executed only if approved (execution instant or delayed)
- There’s some kind of governance token involved (UNI, COMP, MKR, etc.)

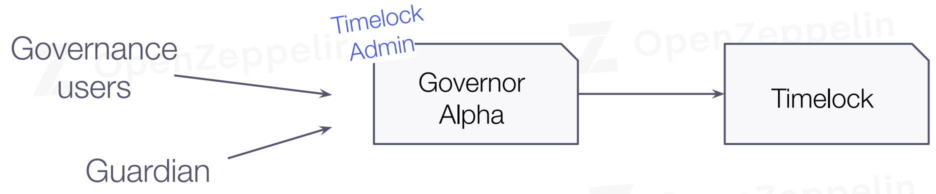
governance

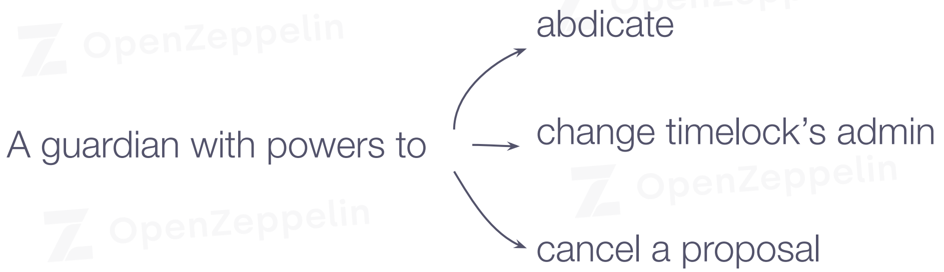


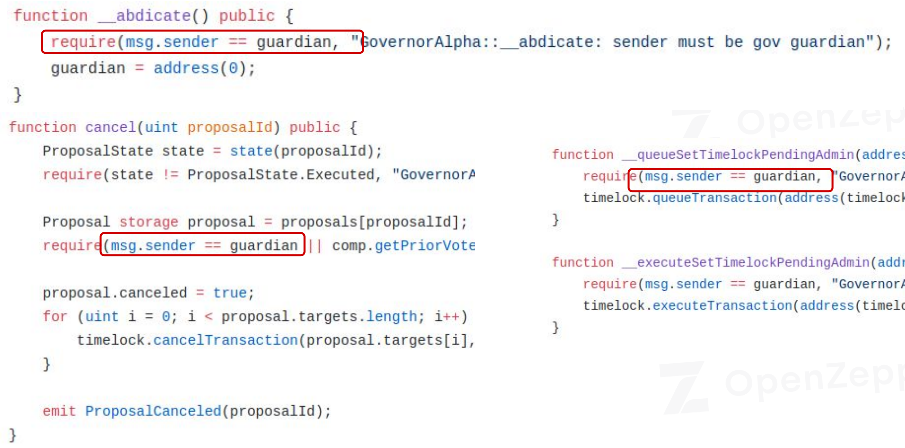
Uniswap is quite similar, without guardian

A guardian with powers to
- abdicate
- change timelock's admin
- cancel a proposal

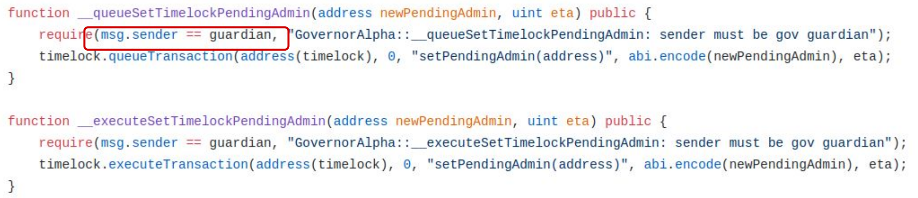
governance







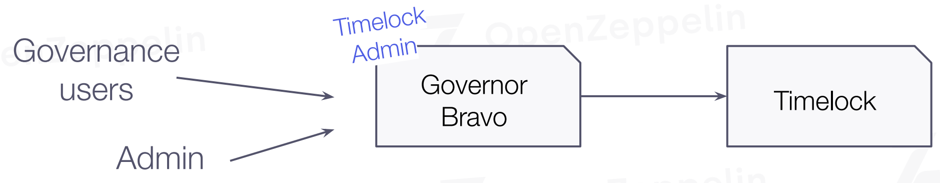
```
function __abdicate() public {
    require(msg.sender == guardian, "GovernorAlpha::__abdicate: sender must be gov guardian");
     guardian = address(0);
 }

function cancel(uint proposalId) public {
    ProposalState state = state(proposalId);
    require(state != ProposalState.Executed, "GovernorA

    Proposal storage proposal = proposals[proposalId];
    require(msg.sender == guardian || comp.getPriorVote

    proposal.canceled = true;
    for (uint i = 0; i < proposal.targets.length; i++)
        timelock.cancelTransaction(proposal.targets[i],
    }

    emit ProposalCanceled(proposalId);
}
```



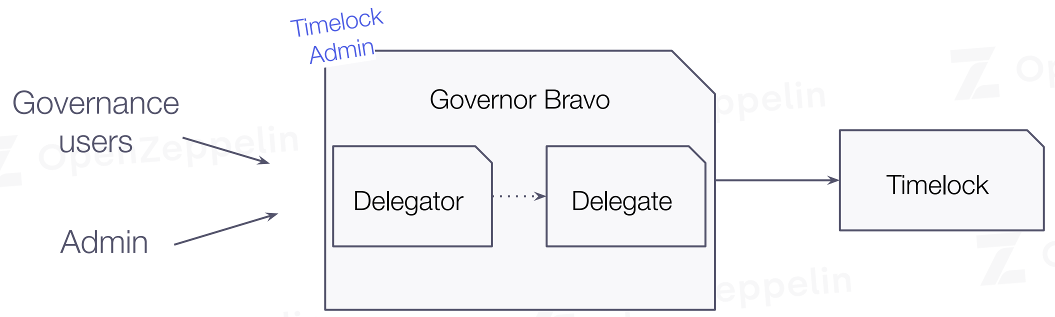
```
function __queueSetTimelockPendingAdmin(address newPendingAdmin, uint eta) public {
    require(msg.sender == guardian, "GovernorAlpha::__queueSetTimelockPendingAdmin: sender must be gov guardian");
    timelock.queueTransaction(address(timelock), 0, "setPendingAdmin(address)", abi.encode(newPendingAdmin), eta);
}

function __executeSetTimelockPendingAdmin(address newPendingAdmin, uint eta) public {
    require(msg.sender == guardian, "GovernorAlpha::__executeSetTimelockPendingAdmin: sender must be gov guardian");
    timelock.executeTransaction(address(timelock), 0, "setPendingAdmin(address)", abi.encode(newPendingAdmin), eta);
}
```

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

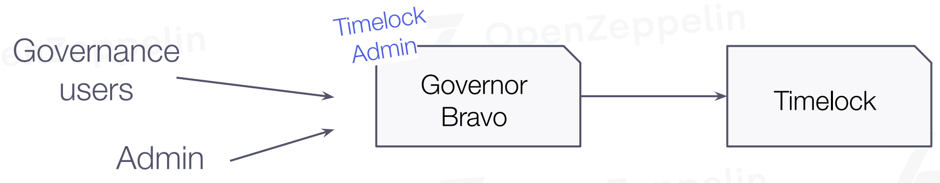
governance

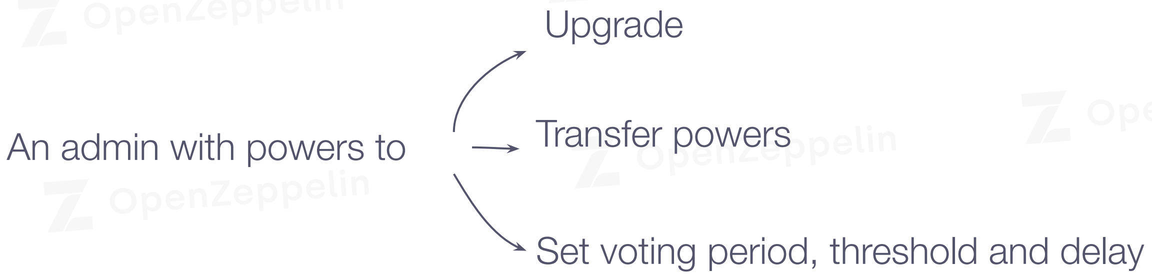

Governance users
Admin
Timelock Admin
Governor Bravo
Timelock

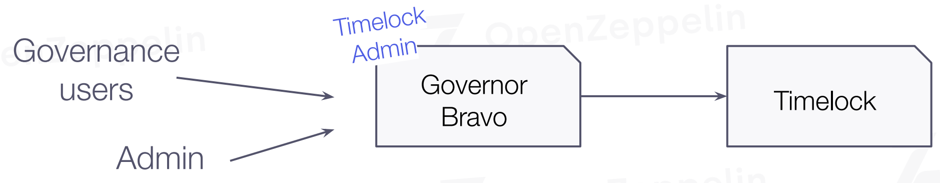
governance


Timelock
Admin
Governor Bravo
Governance
users
Admin
Delegator
Delegate
Timelock

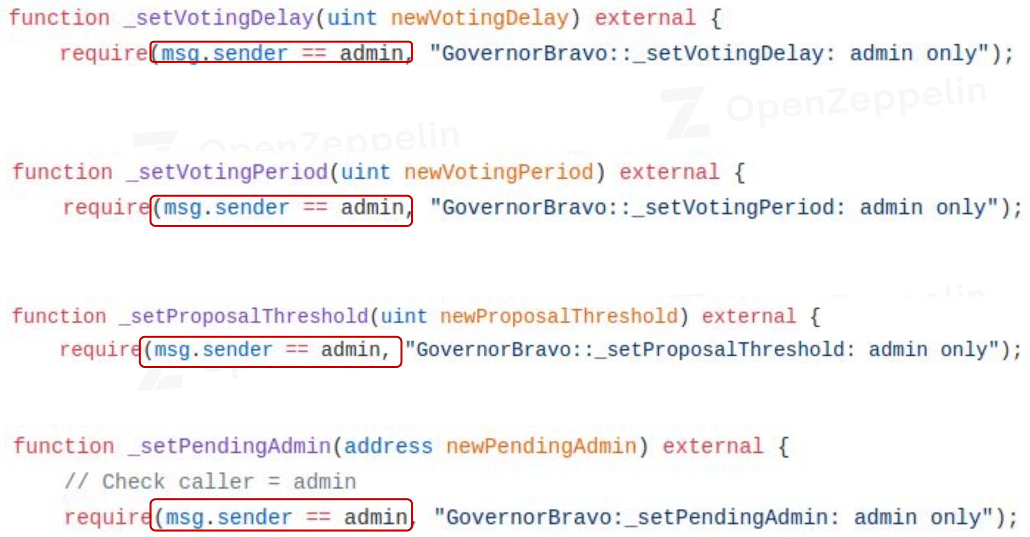
governance


Governance users
Admin
Timelock Admin
Governor Bravo
Timelock


An admin with powers to
Upgrade
Transfer powers
Set voting period, threshold and delay

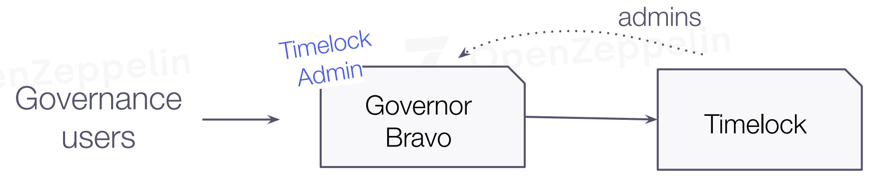


governance



```
function _setVotingDelay(uint newVotingDelay) external {
    require(msg.sender == admin, "GovernorBravo::_setVotingDelay: admin only");


function _setVotingPeriod(uint newVotingPeriod) external {
    require(msg.sender == admin, "GovernorBravo::_setVotingPeriod: admin only");


function _setProposalThreshold(uint newProposalThreshold) external {
    require(msg.sender == admin, "GovernorBravo::_setProposalThreshold: admin only");


function _setPendingAdmin(address newPendingAdmin) external {
    // Check caller = admin
    require(msg.sender == admin, "GovernorBravo:_setPendingAdmin: admin only");
```

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



governance

An **admin** with powers to

- Upgrade
- Transfer powers
- Set voting period, threshold and delay

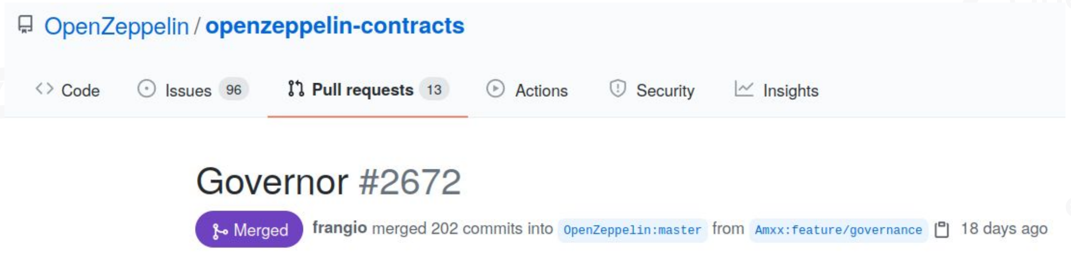
the Timelock!

Changes to governance itself are

also delayed and opt-out

governance

Governance is coming to **OpenZeppelin Contracts**


OpenZeppelin / **openzeppelin-contracts**

Governor #2672

Merged **frangio** merged 202 commits into `OpenZeppelin:master` from `Amxx:feature/governance` 18 days ago

https://github.com/OpenZeppelin/openzeppelin-contracts/pull/2672

https://github.com/OpenZeppelin/openzeppelin-contracts/tree/master/contracts/governance

governance

- Guardians can help
  - Push if there’s no active community participation
  - Stop if there’s malicious intent
- Consider delegation mechanisms of voting power (such as in UNI)
- If governance has full control, off-chain validations, checklists & documentation on procedures becomes crucial.

governance

what about the governance token ?

governance

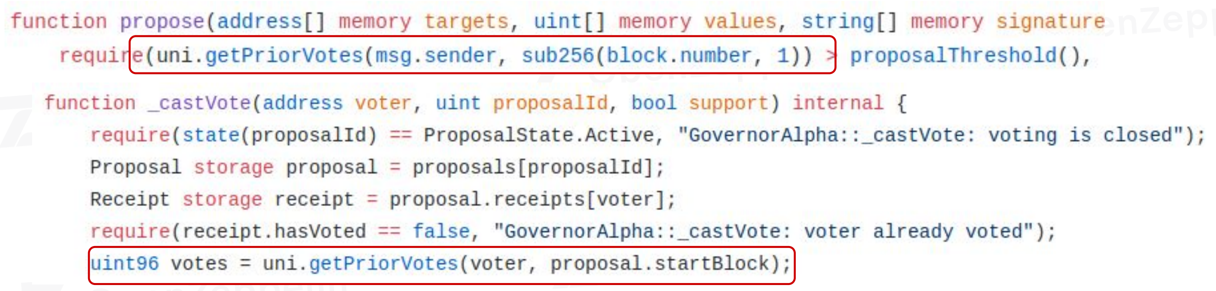
reduce flash loan risk of governance tokens

don’t measure voting power at current block

reducing flash loan risk of governance tokens

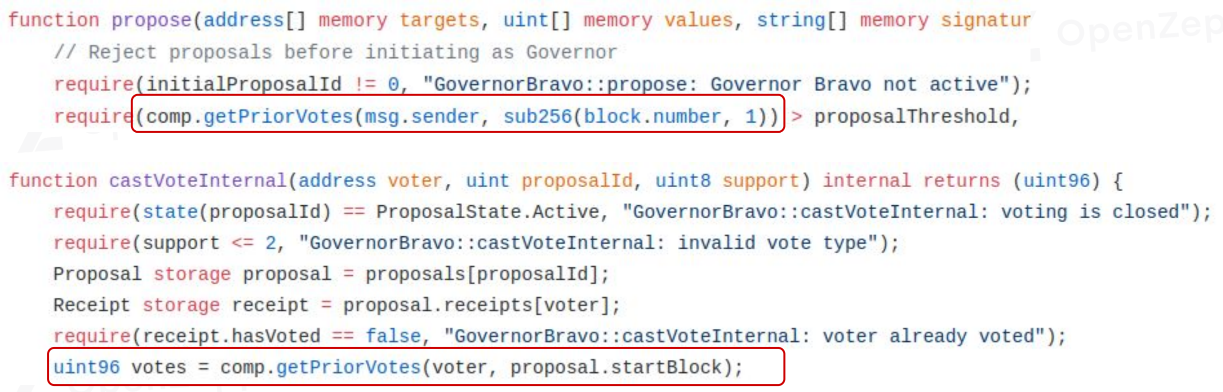
governance

Uniswap GovernorAlpha



```
function propose(address[] memory targets, uint[] memory values, string[] memory signature
    require(uni.getPriorVotes(msg.sender, sub256(block.number, 1)) > proposalThreshold(),

  function _castVote(address voter, uint proposalId, bool support) internal {
      require(state(proposalId) == ProposalState.Active, "GovernorAlpha::_castVote: voting is closed");
      Proposal storage proposal = proposals[proposalId];
      Receipt storage receipt = proposal.receipts[voter];
      require(receipt.hasVoted == false, "GovernorAlpha::_castVote: voter already voted");
      uint96 votes = uni.getPriorVotes(voter, proposal.startBlock);
```

Compound GovernorBravo



```
function propose(address[] memory targets, uint[] memory values, string[] memory signatur
    // Reject proposals before initiating as Governor
    require(initialProposalId != 0, "GovernorBravo::propose: Governor Bravo not active");
    require(comp.getPriorVotes(msg.sender, sub256(block.number, 1)) > proposalThreshold,

function castVoteInternal(address voter, uint proposalId, uint8 support) internal returns (uint96) {
    require(state(proposalId) == ProposalState.Active, "GovernorBravo::castVoteInternal: voting is closed");
    require(support <= 2, "GovernorBravo::castVoteInternal: invalid vote type");
    Proposal storage proposal = proposals[proposalId];
    Receipt storage receipt = proposal.receipts[voter];
    require(receipt.hasVoted == false, "GovernorBravo::castVoteInternal: voter already voted");
    uint96 votes = comp.getPriorVotes(voter, proposal.startBlock);
```

On secure access controls

Closing thoughts

1 Consistent and reliable access controls.

2 Progressive decentralization approach. Share with community.

3 Document all roles in your system. Be transparent.

4 Take advantage of battle-tested and secure building blocks.

5 Start small and focused. Learn from others.

On access controls

Where do I learn more ?

- ➔ Access Control in docs.openzeppelin.com/contracts
- ➔ Multisigs and key management in Defender Advisor
- ➔ Defender Relayers in

  docs.openzeppelin.com/defender/relay
- ➔ ethereum.org/en/wallets

Series of sessions

Secure Development

- The dangers of token integration ✓
- Strategies for secure access controls ✓
- The dangers of price oracles
- and more!

In the meantime…

docs.openzeppelin.com/defender/advisor

We're hiring!

Open Roles

- Blockchain Security Engineer
- Full Stack Ethereum Developer
- Technical Recruiter
- People Ops

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