11/16 - 12PM PST / 7PM UTC

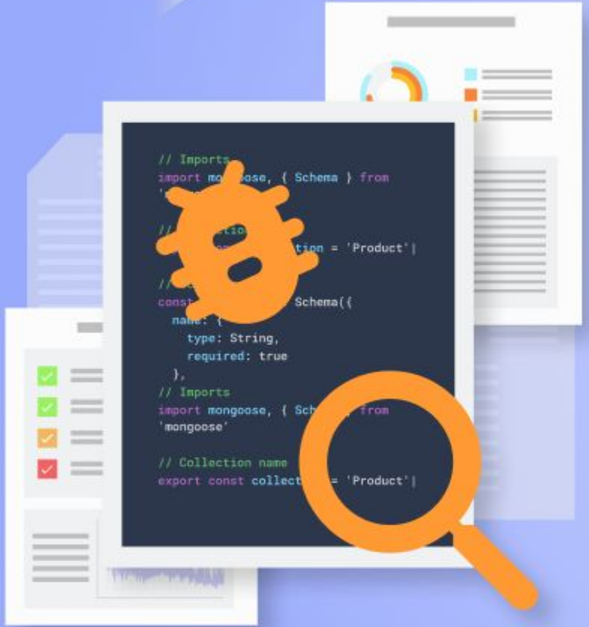
Workshop #6

Secure Development by OpenZeppelin

Onward with Smart Contract Security

Martin Abbatemarco

Security Researcher at OpenZeppelin



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 EXCHANGE

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		

the most concerning issues ?

Integration with tokens

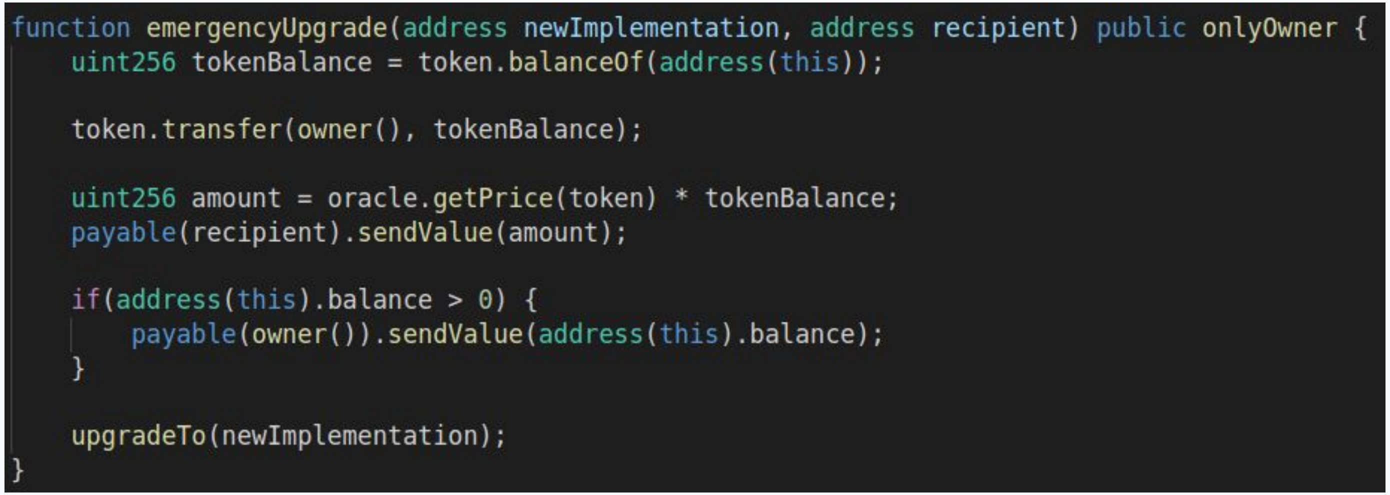
Access controls

Integrations with price oracles

Governance mechanisms

Upgradeability

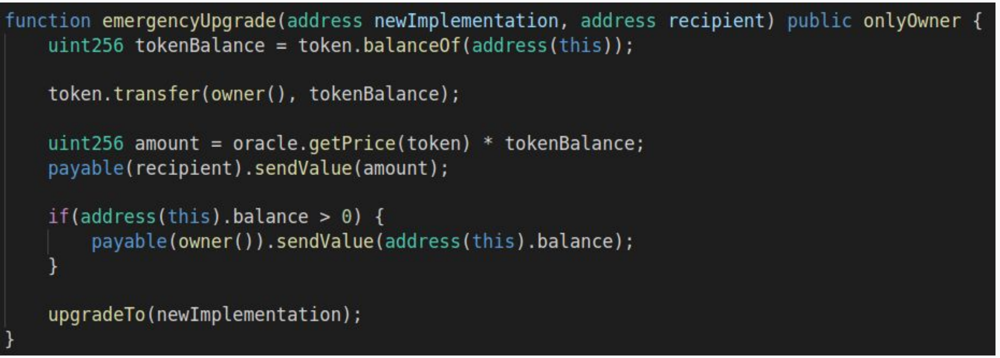
in our previous session



```
function emergencyUpgrade(address newImplementation, address recipient) public onlyOwner {
    uint256 tokenBalance = token.balanceOf(address(this));

    token.transfer(owner(), tokenBalance);

    uint256 amount = oracle.getPrice(token) * tokenBalance;
    payable(recipient).sendValue(amount);

    if(address(this).balance > 0) {
        payable(owner()).sendValue(address(this).balance);
    }

    upgradeTo(newImplementation);
}
```

question beyond the code

question beyond the code



```
function emergencyUpgrade(address newImplementation, address recipient) public onlyOwner {
    uint256 tokenBalance = token.balanceOf(address(this));

    token.transfer(owner(), tokenBalance);

    uint256 amount = oracle.getPrice(token) * tokenBalance;
    payable(recipient).sendValue(amount);

    if(address(this).balance > 0) {
        payable(owner()).sendValue(address(this).balance);
    }

    upgradeTo(newImplementation);
}
```

Available documentation ?

Is this even tested ?

How are others implementing similar features?

How is this going to be launched ?

How is this going to be operated ?

Is this going to be peer-reviewed and audited ?

Bug bounties ?

Monitoring systems in place ?

what else contributes to security ?

taking a step back

Starting small & focused

Starting small and focused

- Beware, there's *lots* going on.
- Define interactions and interfaces before coding.
- Consider design documents and specs. Involve non-developers.
- Reuse what's at hand.

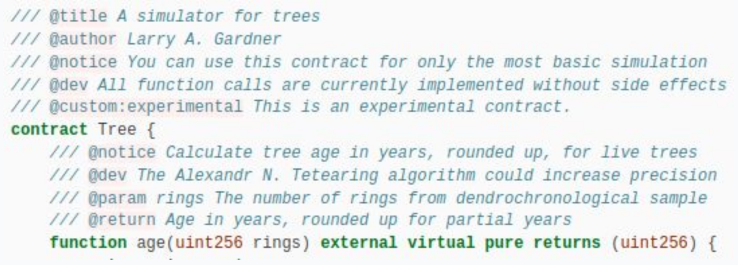
Documentation

intention

implementation

- Inline comments
- Rich documentation → contracts, functions, variables. NatSpec format.

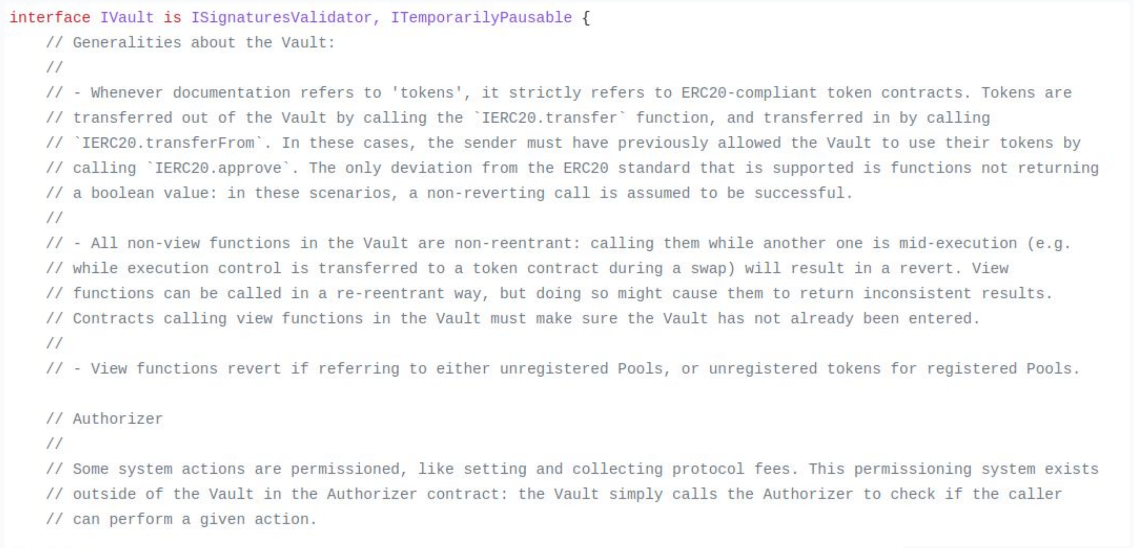
documentation for intention



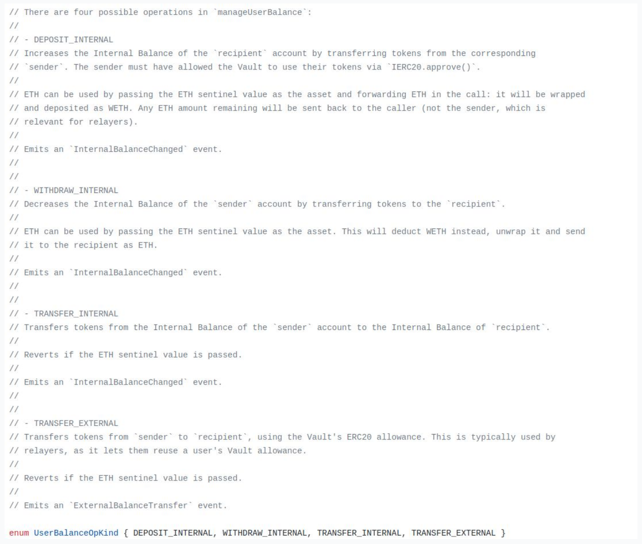
```
/// @title A simulator for trees
/// @author Larry A. Gardner
/// @notice You can use this contract for only the most basic simulation
/// @dev All function calls are currently implemented without side effects
/// @custom:experimental This is an experimental contract.
contract Tree {
    /// @notice Calculate tree age in years, rounded up, for live trees
    /// @dev The Alexandr N. Tetearing algorithm could increase precision
    /// @param rings The number of rings from dendrochronological sample
    /// @return Age in years, rounded up for partial years
    function age(uint256 rings) external virtual pure returns (uint256) {
```

- Inline comments
- Rich documentation → contracts, functions, variables. NatSpec format.

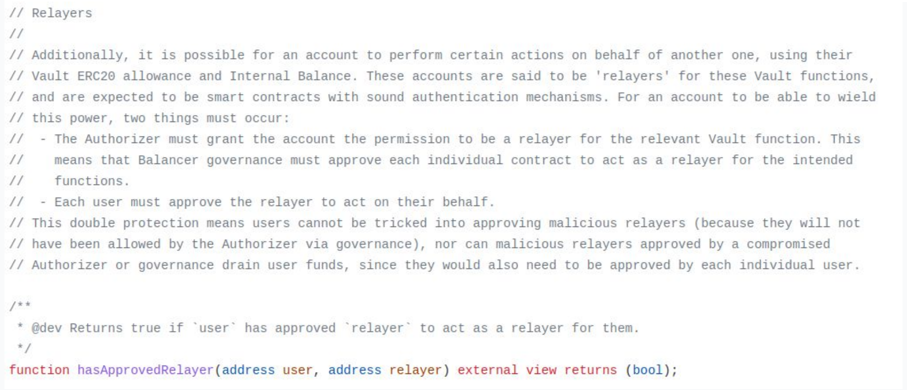
documentation for intention



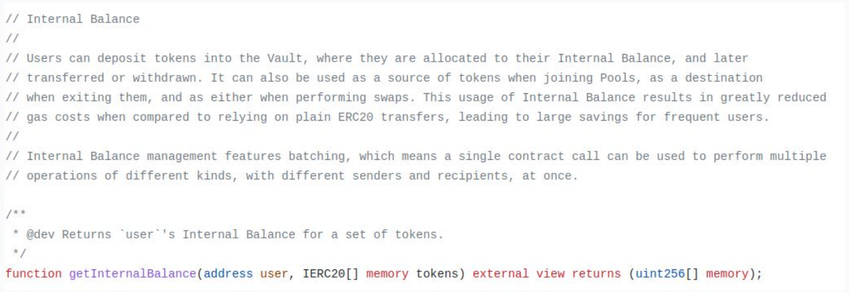
```solidity
interface IVault is ISignaturesValidator, ITemporarilyPausable {
    // Generalities about the Vault:
    //
    // - Whenever documentation refers to 'tokens', it strictly refers to ERC20-compliant token contracts. Tokens are
    // transferred out of the Vault by calling the `IERC20.transfer` function, and transferred in by calling
    // `IERC20.transferFrom`. In these cases, the sender must have previously allowed the Vault to use their tokens by
    // calling `IERC20.approve`. The only deviation from the ERC20 standard that is supported is functions not returning
    // a boolean value: in these scenarios, a non-reverting call is assumed to be successful.
    //
    // - All non-view functions in the Vault are non-reentrant: calling them while another one is mid-execution (e.g.
    // while execution control is transferred to a token contract during a swap) will result in a revert. View
    // functions can be called in a re-reentrant way, but doing so might cause them to return inconsistent results.
    // Contracts calling view functions in the Vault must make sure the Vault has not already been entered.
    //
    // - View functions revert if referring to either unregistered Pools, or unregistered tokens for registered Pools.

    // Authorizer
    //
    // Some system actions are permissioned, like setting and collecting protocol fees. This permissioning system exists
    // outside of the Vault in the Authorizer contract: the Vault simply calls the Authorizer to check if the caller
    // can perform a given action.
```



```solidity
// There are four possible operations in `manageUserBalance`:
//
// - DEPOSIT_INTERNAL
// Increases the Internal Balance of the `recipient` account by transferring tokens from the corresponding
// `sender`. The sender must have allowed the Vault to use their tokens via `IERC20.approve()`.
//
// ETH can be used by passing the ETH sentinel value as the asset and forwarding ETH in the call: it will be wrapped
// and deposited as WETH. Any ETH amount remaining will be sent back to the caller (not the sender, which is
// relevant for relayers).
//
// Emits an `InternalBalanceChanged` event.
//
//
// - WITHDRAW_INTERNAL
// Decreases the Internal Balance of the `sender` account by transferring tokens to the `recipient`.
//
// ETH can be used by passing the ETH sentinel value as the asset. This will deduct WETH instead, unwrap it and send
// it to the recipient as ETH.
//
// Emits an `InternalBalanceChanged` event.
//
//
// - TRANSFER_INTERNAL
// Transfers tokens from the Internal Balance of the `sender` account to the Internal Balance of `recipient`.
//
// Reverts if the ETH sentinel value is passed.
//
// Emits an `InternalBalanceChanged` event.
//
//
// - TRANSFER_EXTERNAL
// Transfers tokens from `sender` to `recipient`, using the Vault's ERC20 allowance. This is typically used by
// relayers, as it lets them reuse a user's Vault allowance.
//
// Reverts if the ETH sentinel value is passed.
//
// Emits an `ExternalBalanceTransfer` event.

enum UserBalanceOpKind { DEPOSIT_INTERNAL, WITHDRAW_INTERNAL, TRANSFER_INTERNAL, TRANSFER_EXTERNAL }
```

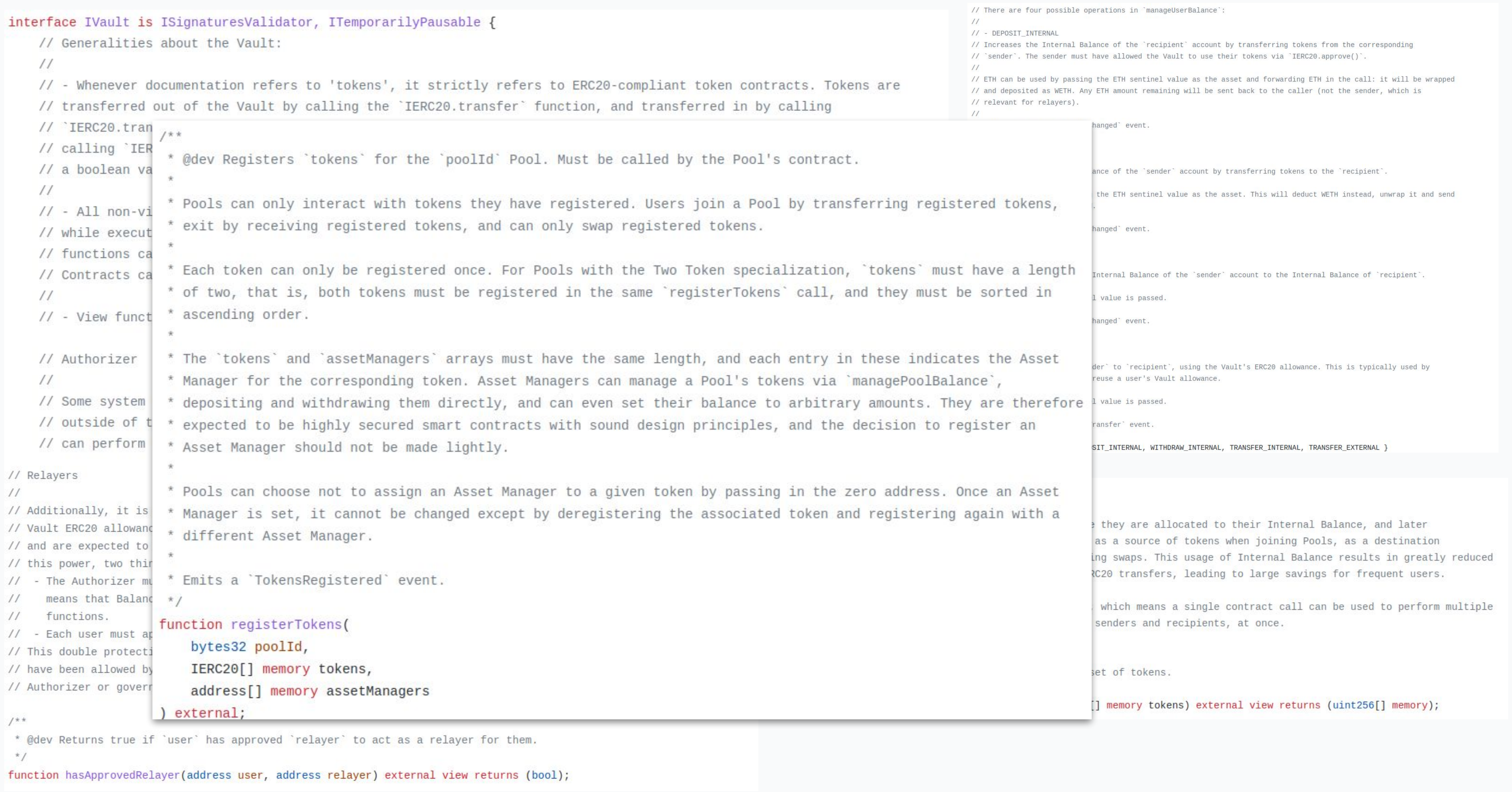


```solidity
// Relayers
//
// Additionally, it is possible for an account to perform certain actions on behalf of another one, using their
// Vault ERC20 allowance and Internal Balance. These accounts are said to be 'relayers' for these Vault functions,
// and are expected to be smart contracts with sound authentication mechanisms. For an account to be able to wield
// this power, two things must occur:
//  - The Authorizer must grant the account the permission to be a relayer for the relevant Vault function. This
//    means that Balancer governance must approve each individual contract to act as a relayer for the intended
//    functions.
//  - Each user must approve the relayer to act on their behalf.
// This double protection means users cannot be tricked into approving malicious relayers (because they will not
// have been allowed by the Authorizer via governance), nor can malicious relayers approved by a compromised
// Authorizer or governance drain user funds, since they would also need to be approved by each individual user.

/**
 * @dev Returns true if `user` has approved `relayer` to act as a relayer for them.
 */
function hasApprovedRelayer(address user, address relayer) external view returns (bool);
```



```solidity
// Internal Balance
//
// Users can deposit tokens into the Vault, where they are allocated to their Internal Balance, and later
// transferred or withdrawn. It can also be used as a source of tokens when joining Pools, as a destination
// when exiting them, and as either when performing swaps. This usage of Internal Balance results in greatly reduced
// gas costs when compared to relying on plain ERC20 transfers, leading to large savings for frequent users.
//
// Internal Balance management features batching, which means a single contract call can be used to perform multiple
// operations of different kinds, with different senders and recipients, at once.

/**
 * @dev Returns `user`'s Internal Balance for a set of tokens.
 */
function getInternalBalance(address user, IERC20[] memory tokens) external view returns (uint256[] memory);
```

https://github.com/balancer-labs/balancer-v2-monorepo/blob/master/pkg/vault/contracts/interfaces/IVault.sol



```
interface IVault is ISignaturesValidator, ITemporarilyPausable {
    // Generalities about the Vault:
    //
    // - Whenever documentation refers to 'tokens', it strictly refers to ERC20-compliant token contracts. Tokens are
    // transferred out of the Vault by calling the `IERC20.transfer` function, and transferred in by calling
    // `IERC20.tran
    // calling `IER
    // a boolean va
    //
    // - All non-vi
    // while execut
    // functions ca
    // Contracts ca
    //
    // - View funct

    // Authorizer
    //
    // Some system
    // outside of t
    // can perform

// Relayers
//
// Additionally, it is
// Vault ERC20 allowanc
// and are expected to
// this power, two thi
//  - The Authorizer mu
//    means that Balanc
//    functions.
//  - Each user must ap
// This double protecti
// have been allowed by
// Authorizer or gover

/**
 * @dev Returns true if `user` has approved `relayer` to act as a relayer for them.
 */
function hasApprovedRelayer(address user, address relayer) external view returns (bool);
```

```
// There are four possible operations in `manageUserBalance`:
//
// - DEPOSIT_INTERNAL
// Increases the Internal Balance of the `recipient` account by transferring tokens from the corresponding
// `sender`. The sender must have allowed the Vault to use their tokens via `IERC20.approve()`.
//
// ETH can be used by passing the ETH sentinel value as the asset and forwarding ETH in the call: it will be wrapped
// and deposited as WETH. Any ETH amount remaining will be sent back to the caller (not the sender, which is
// relevant for relayers).
//
hanged` event.

ance of the `sender` account by transferring tokens to the `recipient`.

 the ETH sentinel value as the asset. This will deduct WETH instead, unwrap it and send
.

hanged` event.

Internal Balance of the `sender` account to the Internal Balance of `recipient`.

l value is passed.

hanged` event.

der` to `recipient`, using the Vault's ERC20 allowance. This is typically used by
reuse a user's Vault allowance.

l value is passed.

ransfer` event.

SIT_INTERNAL, WITHDRAW_INTERNAL, TRANSFER_INTERNAL, TRANSFER_EXTERNAL }
```

```
e they are allocated to their Internal Balance, and later
 as a source of tokens when joining Pools, as a destination
ing swaps. This usage of Internal Balance results in greatly reduced
RC20 transfers, leading to large savings for frequent users.

 which means a single contract call can be used to perform multiple
 senders and recipients, at once.

set of tokens.

] memory tokens) external view returns (uint256[] memory);
```

```
/**
 * @dev Registers `tokens` for the `poolId` Pool. Must be called by the Pool's contract.
 *
 * Pools can only interact with tokens they have registered. Users join a Pool by transferring registered tokens,
 * exit by receiving registered tokens, and can only swap registered tokens.
 *
 * Each token can only be registered once. For Pools with the Two Token specialization, `tokens` must have a length
 * of two, that is, both tokens must be registered in the same `registerTokens` call, and they must be sorted in
 * ascending order.
 *
 * The `tokens` and `assetManagers` arrays must have the same length, and each entry in these indicates the Asset
 * Manager for the corresponding token. Asset Managers can manage a Pool's tokens via `managePoolBalance`,
 * depositing and withdrawing them directly, and can even set their balance to arbitrary amounts. They are therefore
 * expected to be highly secured smart contracts with sound design principles, and the decision to register an
 * Asset Manager should not be made lightly.
 *
 * Pools can choose not to assign an Asset Manager to a given token by passing in the zero address. Once an Asset
 * Manager is set, it cannot be changed except by deregistering the associated token and registering again with a
 * different Asset Manager.
 *
 * Emits a `TokensRegistered` event.
 */
function registerTokens(
    bytes32 poolId,
    IERC20[] memory tokens,
    address[] memory assetManagers
) external;
```

https://github.com/balancer-labs/balancer-v2-monorepo/blob/master/pkg/vault/contracts/interfaces/IVault.sol

documentation for intention

- Inline comments
- Rich documentation → contracts, functions, variables. NatSpec format.
- Tools → https://github.com/OpenZeppelin/solidity-docgen
- External documentation
    - architecture, economic incentives, roles, design decisions, expected use cases, failure modes, integrations, adversarial scenarios, etc.
- README files

Testing

considerations for testing

- Speed matters
- Test for edge cases
- Test against local forks
- Aim for high coverage
- Invest in helpers
- Fuzzing, property testing, symbolic execution
- Follow others → review testing suites, read guides

https://github.com/MolochVentures/moloch/tree/master/test

Learning from others

from **success**

diving into codebases

how does X do Y ?

share insights

don’t blindly copy-paste :)

from **mistakes**

read security newsletters

analyze public security issues

play wargames and CTFs

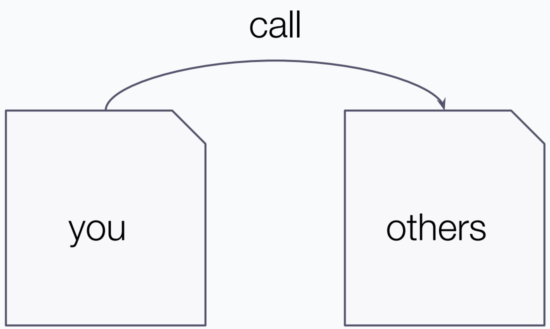
read public audit reports

Risks of integrations

risks of integrations

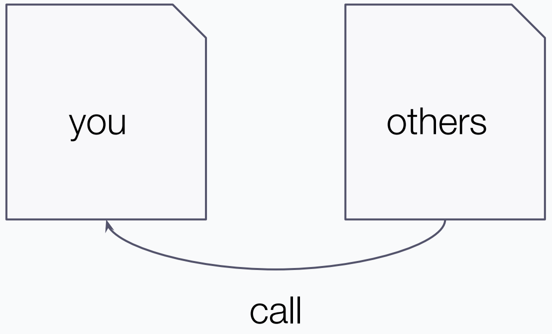
- Calling others
- Others calling you
- Economic risks, price manipulations
- Cross-domain integration

risks of integrations



- Healthy distrust for “standards”
    - Don’t blindly trust an interface. Verify behavior.
- Don’t assume immutability
- Think of threats introduced by integration
- Beware of calls and delegatecalls to user-supplied address

risks of integrations



- Remember access controls :)
- Assume multi-step operations in unusual order
- Anybody can be a whale
- Frontend protections are mostly useless
- Unverified source code won’t prevent much

risks of integrations

- Calling others
- Others calling you
- **Economic risks and price manipulations**
- Cross-domain integration

risks of integrations

- Calling others
- Others calling you
- Economic risks and price manipulations
- **Cross-domain integration**
    - What you call (or what calls you) might not be in your domain

Launching with safeguards

Launching with safeguards

- Permissioned systems. Beware with too many roles
- Pausing, freezing, emergency shutdowns
- Upgradeability mechanisms
- Caps in the amount of funds
- Integrations with battle-tested protocols / tokens / oracles
- Monitoring most critical operations

Launching with safeguards

- Incident response plans
- Security contacts
- Bug bounties and contests
- Core and experimental features
- Beware of public exposure

Importance and limitations of audits

Importance & limitations of audits

Necessary, not sufficient

Importance & limitations of audits

One-off audits --> continuous engagements

Importance & limitations of audits

Can fail, will fail.

Better aim for **defense in depth**.

$\swarrow$

Insurance, monitoring infrastructure,

incident response, recovery plans

Importance & limitations of audits

More complements

- security-first dev mindset
- contests
- bug bounties
- automated security tooling
- formal verification

Monitoring

Are you being hacked right now ?

monitoring

What ?

- drop in system funds
- spikes in account activity, function calls
- privileged admin actions
- significant price changes
- large value txs, high gas txs
- sandwich attacks affecting your users

but how ?

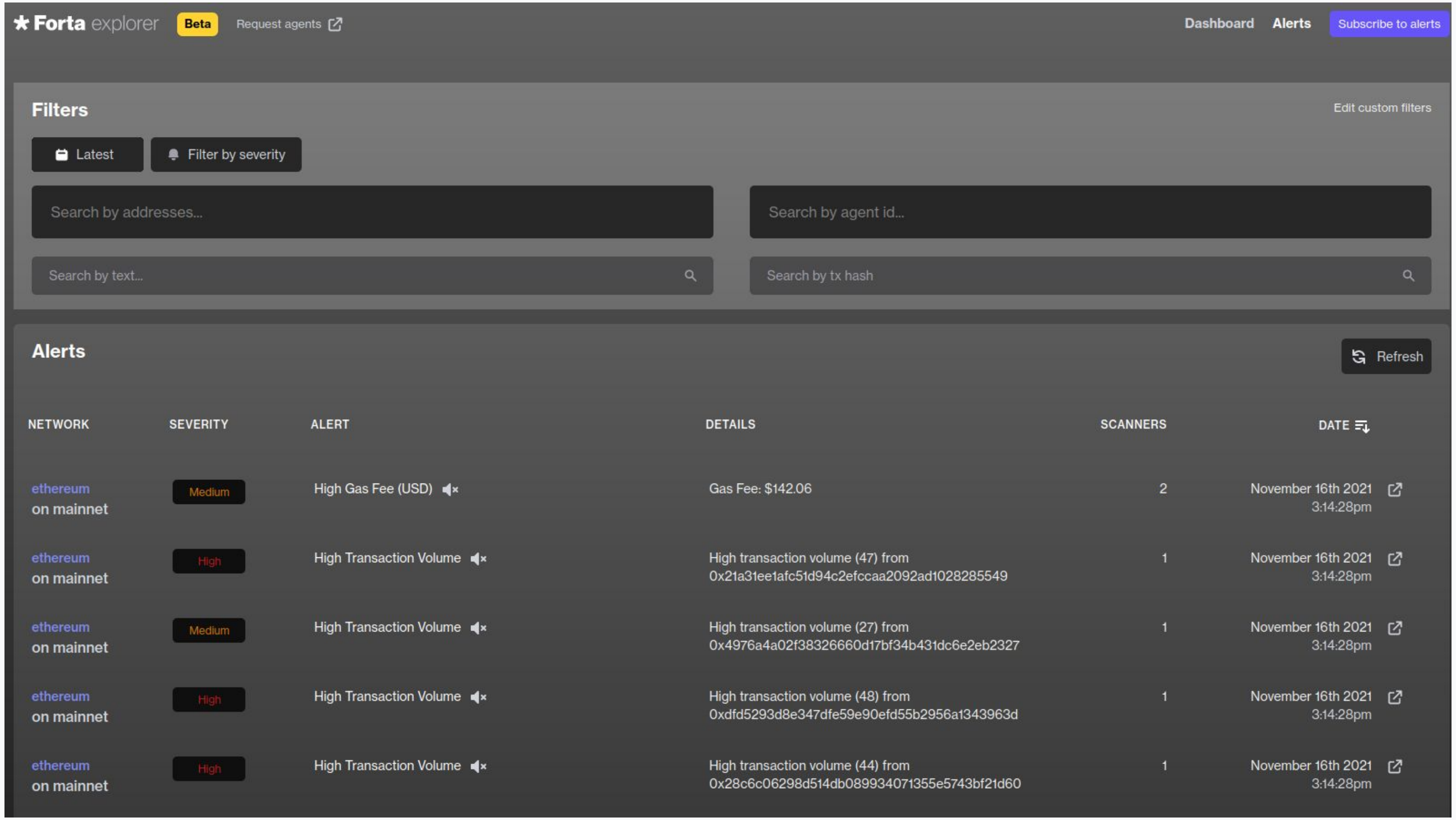
monitoring

- OpenZeppelin Defender (openzeppelin.com/defender)
- Forta agents (forta.org)
- Tenderly (tenderly.co)
- Blocknative (blocknative.com)
- The Graph (thegraph.com)

monitoring

- **OpenZeppelin Defender** (openzeppelin.com/defender)
- **Forta agents** (forta.org)
- Tenderly (tenderly.co)
- Blocknative (blocknative.com)
- The Graph (thegraph.com)

explorer.forta.network



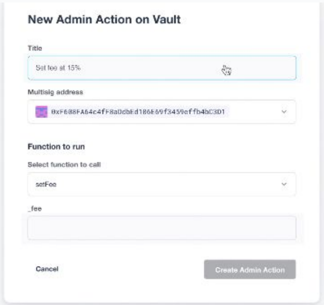
NETWORK	SEVERITY	ALERT	DETAILS	SCANNERS	DATE
ethereum on mainnet	Medium	High Gas Fee (USD)	Gas Fee: $142.06	2	November 16th 2021 3:14:28pm
ethereum on mainnet	High	High Transaction Volume	High transaction volume (47) from 0x21a31ee1afc51d94c2efccaa2092ad1028285549	1	November 16th 2021 3:14:28pm
ethereum on mainnet	Medium	High Transaction Volume	High transaction volume (27) from 0x4976a4a02f38326660d17bf34b431dc6e2eb2327	1	November 16th 2021 3:14:28pm
ethereum on mainnet	High	High Transaction Volume	High transaction volume (48) from 0xdfd5293d8e347dfe59e90efd55b2956a1343963d	1	November 16th 2021 3:14:28pm
ethereum on mainnet	High	High Transaction Volume	High transaction volume (44) from 0x28c6c06298d514db089934071355e5743bf21d60	1	November 16th 2021 3:14:28pm

Secure operations

Admin

Automate and secure all your smart contract administration

Administration mistakes on protocols and applications put user funds at risk. With Defender Admin, you can seamlessly manage all smart contract administration including access controls, upgrades, and pausing. Works with popular multi-sigs including Gnosis Safe.



New Admin Action on Vault

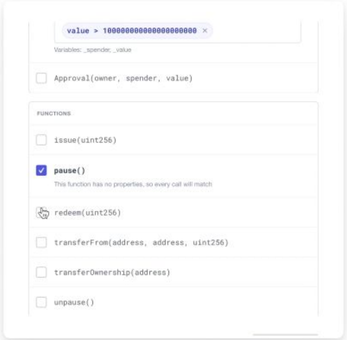
Title: Set fee at 15%

Multisig address: 0xF688FA64c4fF8aDdbEd186E69f3459effb4bC3D1

Function to run

Select function to call: setFee

_fee:



value > 100000000000000000000

Variables: _spender, _value

☐ Approval(owner, spender, value)

FUNCTIONS

☐ issue(uint256)

✓ pause()

This function has no properties, so every call will match

☐ redeem(uint256)

☐ transferFrom(address, address, uint256)

☐ transferOwnership(address)

☐ unpause()

Sentinels

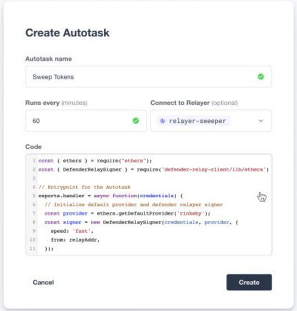
Monitor and respond to smart contract exploits

Use Defender Sentinels to automatically monitor and respond to events, functions, and transaction parameters on your smart contracts. With full Autotask integration, you can add circuit breakers or automated actions so your team can respond to attacks within seconds and receive notifications via email, Slack, Telegram, or Discord.

Autotasks

Create automated scripts to call your smart contracts

Homegrown bots and cron jobs are tedious to maintain and a target for hackers. With Defender Autotasks, you can easily create and run scripts in a serverless environment that call your smart contracts and other web services. Automate your operations and lower attack risk.



Create Autotask

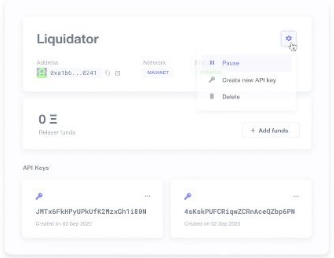
Autotask name: Sweep Tokens

Runs every (minutes): 60

Connect to Relayer (optional): relayer-sweeper

Code

```
const { ethers } = require("ethers");
const { DefenderRelaySigner } = require('defender-relay-client/lib/ethers')

// Entrypoint for the Autotask
exports.handler = async function(credentials) {
  // Initialize default provider and defender relayer signer
  const provider = ethers.getDefaultProvider('rinkeby');
  const signer = new DefenderRelaySigner(credentials, provider, {
    speed: 'fast',
    from: relayAddr,
  });
```



Liquidator

Address: 0xa1b6...8241

Network: MAINNET

0 Ξ

Relayer funds

API Keys

JMTx6FkHPyUPkUfK2MzxGh1i80N

Created on 02 Sep 2020

4sKskPUFCR1qwZCRnAceQZbp6PN

Created on 02 Sep 2020

Relay

Build with private and secure transaction infrastructure

Don't spend time implementing third-party or homegrown transaction infrastructure that is unreliable or insecure. Use Defender Relay to quickly implement private relayers with support for testnets, mainnet, layer 2 and sidechains. Increase user security with embedded key vaults, API key management, and meta-transactions.

openzeppelin.com/defender

Some closing thoughts **from auditors themselves**

With love, from the OpenZeppelin audits team

Code slowly

Assume your users wont use your contracts as you want them to be used

Have a plan for key management (multi-sig arrangements) for admin accounts

Encourage them to use Defender

Read about other DeFi hacks - they open your eyes to what attacks are possible

Get someone to break your contracts as early as possible (like bug bounties)

Don't reinvent the wheel but do know how it works

Launch slowly, first deploying in testnets

Audits are not enough

Set up monitoring systems

Decentralize governance slowly

The importance of testing, auditing & bug bounties

Documentation is key for others to analyze your code

Security is worth investing in!

Encourage your community to know the limitations of your code and security knowledge. Educate them on importance of "slow and steady" deployments

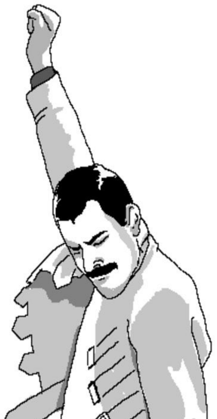
Design first, code later.

Play Damn Vulnerable DeFi and Ethernaut

Using OZ contracts whenever possible

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	✓

Series of sessions

Secure Development

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

youtube.com/openzeppelin

Thanks!

Learn more

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

Contact

@tinchoabbate
tincho@openzeppelin.com