CONSENSYS
Diligence

0x v3 Staking Audit

Date	October 2019
Lead Auditor	Steve Marx
Co-auditors	Alex Wade

1 Summary

ConsenSys Diligence conducted a security audit on the 0x staking contracts. These contracts control the distribution of fees collected by the 0x Exchange to ZRX stakers.

The 0x v3 Exchange audit is good background reading to understand this report.

2 Audit Scope

The scope of this audit was the `staking` project within the 0x monorepo.

This audit covered the following files from commit b8e01d7 of the 0xProject/0x-monorepo:

File Name	SHA-1 Hash
ReadOnlyProxy.sol	6ec64526446ebff87ec5528ee3b2786338cc4fa0
Staking.sol	67ddcb9ab75e433882e28d9186815990b7084c61
StakingProxy.sol	248f562d014d0b1ca6de3212966af3e52a7deef1
ZrxVault.sol	6c3249314868a2f5d0984122e8ab1413a5b521c9
fees/MixinExchangeFees.sol	9ac3b696baa8ba09305cfc83d3c08f17d9d528e1

File Name	SHA-1 Hash
fees/MixinExchangeManager.sol	46f48136a49919cdb5588dc1b3d64c977c3367f2
immutable/MixinConstants.sol	97c2ac83ef97a09cfd485cb0d4b119ba0902cc79
immutable/MixinDeploymentConstants.sol	424f22c45df8e494c4a78f239ea07ff0400d694b
immutable/MixinStorage.sol	8ad475b0e424e7a3ff65eedf2e999cba98f414c8
interfaces/IStaking.sol	ec1d7f214e3fd40e14716de412deee9769359bc0
interfaces/IStakingEvents.sol	25f16b814c4df9d2002316831c3f727d858456c4
interfaces/IStakingProxy.sol	02e35c6b51e08235b2a01d30a8082d60d9d61bee
interfaces/IStorage.sol	eeaa798c262b46d1874e904cf7de0423d4132cee
interfaces/IStorageInit.sol	b9899b03e474ea5adc3b4818a4357f71b8d288d4
interfaces/IStructs.sol	fee17d036883d641afb1222b75eec8427f3cdb96
interfaces/IZrxVault.sol	9067154651675317e000cfa92de9741e50c1c809
libs/LibCobbDouglas.sol	242d62d71cf8bc09177d240c0db59b83f9bb4e96
libs/LibFixedMath.sol	36311e7be09a947fa4e6cd8c544cacd13d65833c
libs/LibFixedMathRichErrors.sol	39cb3e07bbce3272bbf090e87002d5834d288ec2
libs/LibProxy.sol	29abe52857a782c8da39b053cc54e02e295c1ae2
libs/LibSafeDowncast.sol	ae16ed2573d64802793320253b060b9507729c3d
libs/LibStakingRichErrors.sol	f5868ef6066a18277c932e59c0a516ec58920b00
stake/MixinStake.sol	ade59ed356fe72521ffd2ef12ff8896c852f11f8
stake/MixinStakeBalances.sol	cde6ca1a6200570ba18dd6d392ffabf68c2bb464
stake/MixinStakeStorage.sol	cadf34d9d341efd2a85dd13ec3cd4ce8383e0f73
staking_pools/MixinCumulativeRewards.sol	664ea3e35376c81492457dc17832a4d0d602c8ae
staking_pools/MixinStakingPool.sol	74ba9cb2db29b8dd6376d112e9452d117a391b18
staking_pools/MixinStakingPoolRewards.sol	a3b4e5c9b1c3568c94923e2dd9a93090ebdf8536
sys/MixinAbstract.sol	99fd4870c20d8fa03cfa30e8055d3dfb348ed5cd
sys/MixinFinalizer.sol	cc658ed07241c1804cec75b12203be3cd8657b9b
sys/MixinParams.sol	7b395f4da7ed787d7aa4eb915f15377725ff8168
sys/MixinScheduler.sol	2fab6b83a6f9e1d0dd1b1bdcea4b129d166aef1d

The audit activities can be grouped into the following three broad categories:

1. **Security:** Identifying security related issues within the contract.
2. **Architecture:** Evaluating the system architecture through the lens of established smart contract best practices.
3. **Code quality:** A full review of the contract source code. The primary areas of focus include:
    - Correctness
    - Readability
    - Scalability
    - Code complexity
    - Quality of test coverage

3 System Overview

The staking contracts are a mechanism for distributed protocol fees collected by the 0x Exchange. Fees are distributed to pools of ZRX stakeholders according to a formula that takes into account:

1. how much ZRX is being staked by the pool and
2. the amount of protocol fees generated by liquidity providers (“makers”) in that pool.

The v3 staking specification is the best available documentation for understanding how the staking contract system works.

4 Risk Assessment

The code that handles staking is very complex. We remain uncomfortable with parts of the code that were too difficult to audit effectively. That said, this doesn’t mean it’s unsafe to interact with the contract. There are three types of interactions where funds are potentially at risk:

1. ZRX deposits and withdrawals by stakers.
2. The staking contracts hold WETH (wrapped ether) that is collected as protocol fees from the Exchange contracts.
3. Collected WETH is distributed to stakers according to the internal logic of the staking contract.

We can assess the risk associated with all three:

1. ZRX deposits and withdrawals make use of a fairly simple `ZrxVault` contract, which includes a fail-safe mechanism which can be triggered by 0x if needed to allow stakers to directly withdraw their ZRX. Excluding malicious action by 0x themselves, ZRX deposits and withdrawals have low risk of fund loss.
2. Although WETH needs to be approved to the staking contracts, the only WETH actually held by the staking contracts is what is collected in `payProtocolFee`, which is invoked by the Exchange. There's low risk of WETH being inappropriately transferred from users.
3. Most of the complexity of the staking contracts deals with how the collected fees are distributed. This is the part of the code the audit team has less confidence in, meaning there's a relatively higher risk of errors being made here.

This risk assessment means that the most likely type of bug to encounter is one where rewards are paid out incorrectly, or a bug prevents paying out rewards altogether. Those outcomes are no worse for stakers than simply not staking at all.

5 Issues

Each issue has an assigned severity:

- **Minor** issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- **Medium** issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- **Major** issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- **Critical** issues are directly exploitable security vulnerabilities that need to be fixed.

5.1 Anyone can remove a maker's pending pool join status Major ✓ Fixed

Resolution

This is fixed in 0xProject/0x-monorepo#2250 by removing the two-step handshake for a maker to join a pool.

Description

Using behavior described in issue 5.6, it is possible to delete the *pending* join status of *any maker in any pool* by passing in `NIL_POOL_ID` to `removeMakerFromStakingPool`. Note that the attacker in the following example must not be a confirmed member of any pool:

1. The attacker calls `addMakerToStakingPool(NIL_POOL_ID, makerAddress)`. In this case, `makerAddress` can be almost any address, as long as it has not called `joinStakingPoolAsMaker` (an easy example is `address(0)`). The key goal of this call is to increment the number of makers in pool 0:

   code/contracts/staking/contracts/src/staking_pools/MixinStakingPool.sol:L262

   ```
   _poolById[poolId].numberOfMakers = uint256(pool.numberOfMakers).safeAdd(1
   ```

2. The attacker calls `removeMakerFromStakingPool(NIL_POOL_ID, targetAddress)`. This function queries `getStakingPoolIdOfMaker(targetAddress)` and compares it to the passed-in pool id. Because the target is an unconfirmed maker, their staking pool id is `NIL_POOL_ID`:

   code/contracts/staking/contracts/src/staking_pools/MixinStakingPool.sol:L166-L173

   ```
   bytes32 makerPoolId = getStakingPoolIdOfMaker(makerAddress);
   if (makerPoolId != poolId) {
       LibRichErrors.rrevert(LibStakingRichErrors.MakerPoolAssignmentError(
           LibStakingRichErrors.MakerPoolAssignmentErrorCodes.MakerAddressNot
           makerAddress,
           makerPoolId
       ));
   }
   ```

The check passes, and the target's `_poolJoinedByMakerAddress` struct is deleted. Additionally, the number of makers in pool 0 is decreased:

code/contracts/staking/contracts/src/staking_pools/MixinStakingPool.sol:L176-L177

```
delete _poolJoinedByMakerAddress[makerAddress];
_poolById[poolId].numberOfMakers = uint256(_poolById[poolId].numberOfMakers).s
```

This can be used to prevent any makers from being confirmed into a pool.

Recommendation

See issue 5.6.

5.2 Delegated stake weight reduction can be bypassed by using an external contract Major Won't Fix

Resolution

From the development team:

> *Although it is possible to bypass the weight reduction via external smart contracts, we believe there is some value to having a lower delegated stake weight as the default behavior. This can still approximate the intended behavior and should give a very slight edge to pool operators that own their stake.*

Description

Staking pools allow ZRX holders to delegate their staked ZRX to a market maker in exchange for a configurable percentage of the stake reward (accrued over time through exchange fees). When staking as expected through the 0x contracts, the protocol favors ZRX staked directly by the operator of the pool, assigning a lower weight (90%) to ZRX staked by delegation. In return, delegated members receive a configurable portion of the operator’s stake reward.

Using a smart contract, it is possible to represent ZRX owned by any number of parties as ZRX staked by a single party. This contract can serve as the operator of a pool with a single member—itself. The advantages are clear for ZRX holders:

- ZRX staked through this contract will be given full (100%) stake weight.
- Because stake weight is a factor in reward allocation, the ZRX staked through this contract receives a higher proportion of the stake reward.

Recommendation

Remove stake weight reduction for delegated stake.

5.3 `MixinParams.setParams` bypasses safety checks made by standard `StakingProxy` upgrade path. Medium ✓ Fixed

Resolution

This is fixed in 0xProject/0x-monorepo#2279. Now the parameter validity is asserted in `setParams()`.

Description

The staking contracts use a set of configurable parameters to determine the behavior of various parts of the system. The parameters dictate the duration of epochs, the ratio of delegated stake weight vs operator stake, the minimum pool stake, and the Cobb-Douglas numerator and denominator. These parameters can be configured in two ways:

1. An authorized address can deploy a new `Staking` contract (perhaps with altered parameters), and configure the `StakingProxy` to delegate to this new contract. This is done by calling

    - `StakingProxy.detachStakingContract`:

      code/contracts/staking/contracts/src/StakingProxy.sol:L82-L90

      ```
      /// @dev Detach the current staking contract.
      /// Note that this is callable only by an authorized address.
      function detachStakingContract()
          external
          onlyAuthorized
      {
          stakingContract = NIL_ADDRESS;
      ```

```
        emit StakingContractDetachedFromProxy();
    }
```

- `StakingProxy.attachStakingContract(newContract)` :

  code/contracts/staking/contracts/src/StakingProxy.sol:L72-L80

  ```
  /// @dev Attach a staking contract; future calls will be delegated to
  /// Note that this is callable only by an authorized address.
  /// @param _stakingContract Address of staking contract.
  function attachStakingContract(address _stakingContract)
      external
      onlyAuthorized
  {
      _attachStakingContract(_stakingContract);
  }
  ```

During the latter call, the `StakingProxy` performs a delegatecall to `Staking.init` , then checks the values of the parameters set during initialization:

code/contracts/staking/contracts/src/StakingProxy.sol:L208-L219

```
// Call `init()` on the staking contract to initialize storage.
(bool didInitSucceed, bytes memory initReturnData) = stakingContract.deleg
    abi.encodeWithSelector(IStorageInit(0).init.selector)
);
if (!didInitSucceed) {
    assembly {
        revert(add(initReturnData, 0x20), mload(initReturnData))
    }
}

// Assert initialized storage values are valid
_assertValidStorageParams();
```

2. An authorized address can call `MixinParams.setParams` at any time and set the contract’s parameters to arbitrary values.

The latter method introduces the possibility of setting unsafe or nonsensical values for the contract parameters: `epochDurationInSeconds` can be set to 0, `cobbDouglassAlphaNumerator` can be larger than `cobbDouglassAlphaDenominator`, `rewardDelegatedStakeWeight` can be set to a value over 100% of the staking reward, and more.

Note, too, that by using `MixinParams.setParams` to set all parameters to 0, the `Staking` contract can be re-initialized by way of `Staking.init`. Additionally, it can be re-attached by way of `StakingProxy.attachStakingContract`, as the delegatecall to `Staking.init` will succeed.

Recommendation

Ensure that calls to `setParams` check that the provided values are within the same range currently enforced by the proxy.

5.4 Authorized addresses can indefinitely stall `ZrxVaultBackstop` catastrophic failure mode Medium ✓ Fixed

Resolution

This is fixed in 0xProject/0x-monorepo#2295 by removing the `ZrxVaultBackstop` and read-only mode altogether.

Description

The `ZrxVaultBackstop` contract was added to allow anyone to activate the staking system’s “catastrophic failure” mode if the `StakingProxy` is in “read-only” mode for at least 40 days. To enable this behavior, the `StakingProxy` contract was modified to track the last timestamp at which “read-only” mode was activated. This is done by way of `StakingProxy.setReadOnlyMode`:

code/contracts/staking/contracts/src/StakingProxy.sol:L92-L104

```
/// @dev Set read-only mode (state cannot be changed).
function setReadOnlyMode(bool shouldSetReadOnlyMode)
    external
    onlyAuthorized
{
    // solhint-disable-next-line not-rely-on-time
    uint96 timestamp = block.timestamp.downcastToUint96();
    if (shouldSetReadOnlyMode) {
        stakingContract = readOnlyProxy;
        readOnlyState = IStructs.ReadOnlyState({
            isReadOnlyModeSet: true,
            lastSetTimestamp: timestamp
        });
```

Because the timestamp is updated even if “read-only” mode is already active, any authorized address can prevent `ZrxVaultBackstop` from activating catastrophic failure mode by repeatedly calling `setReadOnlyMode`.

Recommendation

If “read-only” mode is already active, `setReadOnlyMode(true)` should result in a no-op.

5.5 Pool 0 can be used to temporarily prevent makers from joining another pool Medium ✓ Fixed

Resolution

This is fixed in 0xProject/0x-monorepo#2250. Pool IDs now start at 1.

Description

`removeMakerFromStakingPool` reverts if the number of makers currently in the pool is 0, due to `safeSub` catching an underflow:

code/contracts/staking/contracts/src/staking_pools/MixinStakingPool.sol:L177

```
_poolById[poolId].numberOfMakers = uint256(_poolById[poolId].numberOfMakers).s
```

Because of this, edge behavior described in issue 5.6 can allow an attacker to temporarily prevent makers from joining a pool:

1. The attacker calls `addMakerToStakingPool(NIL_POOL_ID, victimAddress)`. This sets the victim's `MakerPoolJoinStatus.confirmed` field to `true` and increases the number of makers in pool 0 to 1:

   code/contracts/staking/contracts/src/staking_pools/MixinStakingPool.sol:L257-L262

   ```
   poolJoinStatus = IStructs.MakerPoolJoinStatus({
       poolId: poolId,
       confirmed: true
   });
   _poolJoinedByMakerAddress[makerAddress] = poolJoinStatus;
   _poolById[poolId].numberOfMakers = uint256(pool.numberOfMakers).safeAdd(1
   ```

2. The attacker calls `removeMakerFromStakingPool(NIL_POOL_ID, randomAddress)`. The net effect of this call simply decreases the number of makers in pool 0 by 1, back to 0:

   code/contracts/staking/contracts/src/staking_pools/MixinStakingPool.sol:L176-L177

   ```
   delete _poolJoinedByMakerAddress[makerAddress];
   _poolById[poolId].numberOfMakers = uint256(_poolById[poolId].numberOfMaker
   ```

Typically, the victim should be able to remove themselves from pool 0 by calling `removeMakerFromStakingPool(NIL_POOL_ID, victimAddress)`, but because the attacker can set the pool's number of makers to 0, the aforementioned underflow causes this call to fail. The victim must first understand what is happening in `MixinStakingPool` before they are able to remedy the situation:

1. The victim must call `addMakerToStakingPool(NIL_POOL_ID, randomAddress2)` to increase pool 0's number of makers back to 1.

2. The victim can now call `removeMakerFromStakingPool(NIL_POOL_ID, victimAddress)`, and remove their confirmed status.

Additionally, if the victim in question currently has a pending join, the attacker can use issue 5.1 to first remove their pending status before locking them in pool 0.

Recommendation

See issue 5.1.

5.6 Recommendation: Fix weak assertions in `MixinStakingPool` stemming from use of `NIL_POOL_ID` Medium ✓ Fixed

Resolution

This is fixed in 0xProject/0x-monorepo#2250. Pool IDs now start at 1.

Description

The modifier `onlyStakingPoolOperatorOrMaker(poolId)` is used to authorize actions taken on a given pool. The sender must be either the operator or a confirmed maker of the pool in question. However, the modifier queries `getStakingPoolIdOfMaker(maker)`, which returns `NIL_POOL_ID` if the maker's `MakerPoolJoinStatus` struct is not confirmed. This implicitly makes anyone a maker of the nonexistent "pool 0":

code/contracts/staking/contracts/src/staking_pools/MixinStakingPool.sol:L189-L200

```
function getStakingPoolIdOfMaker(address makerAddress)
    public
    view
    returns (bytes32)
{
    IStructs.MakerPoolJoinStatus memory poolJoinStatus = _poolJoinedByMakerAd
    if (poolJoinStatus.confirmed) {
        return poolJoinStatus.poolId;
    } else {
        return NIL_POOL_ID;
    }
}
```

`joinStakingPoolAsMaker(poolId)` makes no existence checks on the provided pool id, and allows makers to become pending makers in nonexistent pools.

`addMakerToStakingPool(poolId, maker)` makes no existence checks on the provided pool id, allowing makers to be added to nonexistent pools (as long as the sender is an operator or maker in the pool).

Recommendation

1. Avoid use of `0x00...00` for `NIL_POOL_ID`. Instead, use `2**256 - 1`.
2. Implement stronger checks for pool existence. Each time a pool id is supplied, it should be checked that the pool id is between 0 and `nextPoolId`.
3. `onlyStakingPoolOperatorOrMaker` should revert if `poolId == NIL_POOL_ID` or if `poolId` is not in the valid range: (0, nextPoolId).

5.7 `LibFixedMath` functions fail to catch a number of overflows

Medium **✓ Fixed**

Resolution
This is fixed in 0xProject/0x-monorepo#2255 and 0xProject/0x-monorepo#2311.

Description

The `__add()`, `__mul()`, and `__div()` functions perform arithmetic on 256-bit signed integers, and they all miss some specific overflows.

Addition Overflows

code/contracts/staking/contracts/src/libs/LibFixedMath.sol:L359-L376

```
/// @dev Adds two numbers, reverting on overflow.
function _add(int256 a, int256 b) private pure returns (int256 c) {
    c = a + b;
    if (c > 0 && a < 0 && b < 0) {
        LibRichErrors.rrevert(LibFixedMathRichErrors.BinOpError(
            LibFixedMathRichErrors.BinOpErrorCodes.SUBTRACTION_OVERFLOW,
            a,
            b
        ));
    }
    if (c < 0 && a > 0 && b > 0) {
        LibRichErrors.rrevert(LibFixedMathRichErrors.BinOpError(
            LibFixedMathRichErrors.BinOpErrorCodes.ADDITION_OVERFLOW,
            a,
            b
        ));
    }
}
```

The two overflow conditions it tests for are:

1. Adding two positive numbers shouldn't result in a negative number.
2. Adding two negative numbers shouldn't result in a positive number.

`__add(-2**255, -2**255)` returns `0` without reverting because the overflow didn't match either of the above conditions.

Multiplication Overflows

code/contracts/staking/contracts/src/libs/LibFixedMath.sol:L332-L345

```
/// @dev Returns the multiplication two numbers, reverting on overflow.
function _mul(int256 a, int256 b) private pure returns (int256 c) {
    if (a == 0) {
        return 0;
    }
    c = a * b;
    if (c / a != b) {
        LibRichErrors.rrevert(LibFixedMathRichErrors.BinOpError(
            LibFixedMathRichErrors.BinOpErrorCodes.MULTIPLICATION_OVERFLOW,
            a,
            b
        ));
    }
}
```

The function checks via division for most types of overflows, but it fails to catch one particular case. `__mul(-2**255, -1)` returns `-2**255` without error.

Division Overflows

code/contracts/staking/contracts/src/libs/LibFixedMath.sol:L347-L357

```
/// @dev Returns the division of two numbers, reverting on division by zero.
function _div(int256 a, int256 b) private pure returns (int256 c) {
    if (b == 0) {
        LibRichErrors.rrevert(LibFixedMathRichErrors.BinOpError(
            LibFixedMathRichErrors.BinOpErrorCodes.DIVISION_BY_ZERO,
            a,
            b
        ));
    }
    c = a / b;
}
```

It does not check for overflow. Due to this, `__div(-2**255, -1)` erroneously returns `-2**255`.

Recommendation

For addition, the specific case of `__add(-2**255, -2**255)` can be detected by using a `>= 0` check instead of `> 0`, but the below seems like a clearer check for all cases:

```
// if b is negative, then the result should be less than a
if (b < 0 && c >= a) { /* subtraction overflow */ }

// if b is positive, then the result should be greater than a
if (b > 0 && c <= a) { /* addition overflow */ }
```

For multiplication and division, the specific values of `-2**255` and `-1` are the only missing cases, so that can be explicitly checked in the `__mul()` and `__div()` functions.

5.8 Recommendation: Remove `MixinAbstract` and fold `MixinStakingPoolRewards` into `MixinFinalizer` and `MixinStake`

Minor Won't Fix

Resolution

The development team investigated this suggestion, but they were ultimately uncomfortable making such a large change in this cycle. This can be considered again in a future version of the code.

Description

After implementing issue 5.12, issue 5.11, issue 5.10, and issue 5.9, `MixinAbstract` serves little utility except as a way to pull functionality from `MixinFinalizer` into `MixinStakingPoolRewards`. The abstract pattern adds unnecessary cognitive overhead and should be eliminated if possible. One possible method for this is as follows:

1. Move `MixinStakingPoolRewards.withdrawDelegatorRewards` into `MixinStake`. As per the comments above this function, its behavior is very similar to functions in `MixinStake`:

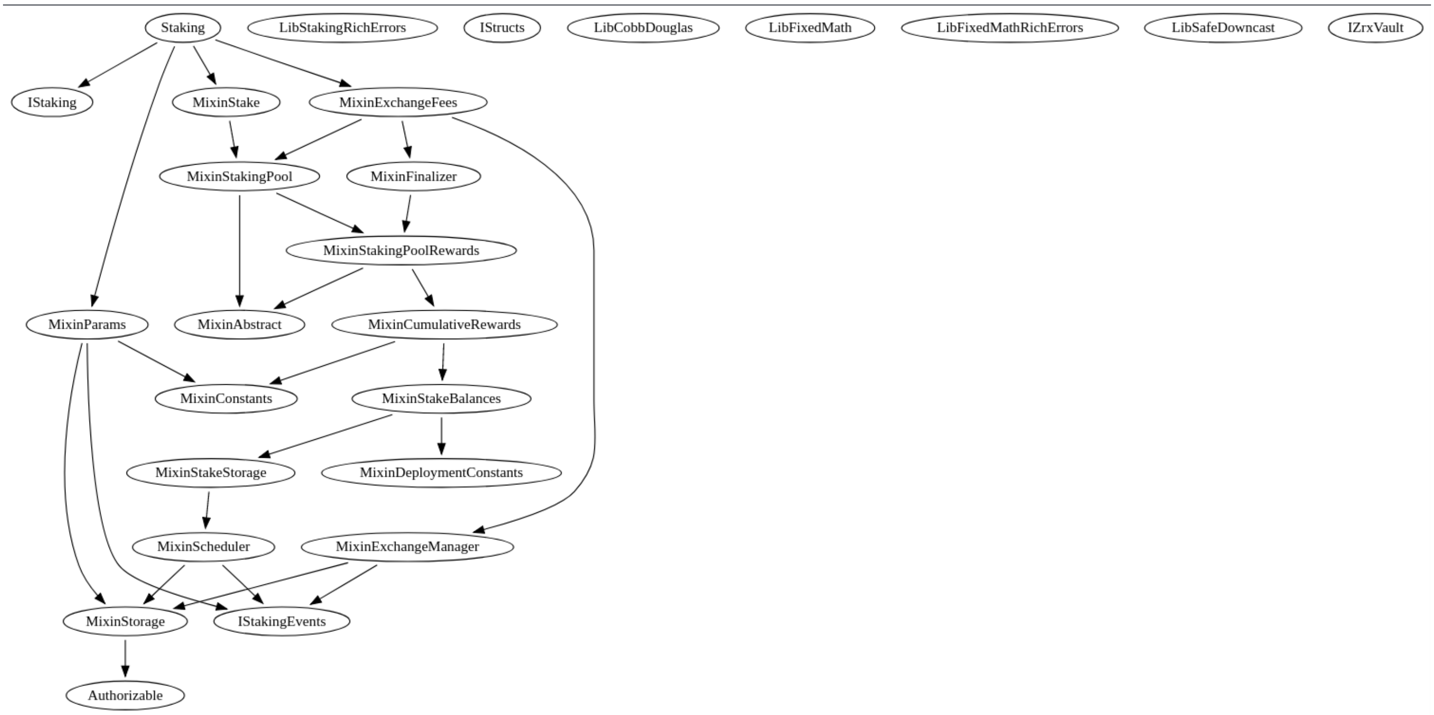
   code/contracts/staking/contracts/src/staking_pools/MixinStakingPoolRewards.sol:L35-L56

   ```
   /// @dev Syncs rewards for a delegator. This includes transferring WETH
   ///      rewards to the delegator, and adding/removing
   ///      dependencies on cumulative rewards.
   ///      This is used by a delegator when they want to sync their rewards
   ///      without delegating/undelegating. It's effectively the same as
   ///      delegating zero stake.
   /// @param poolId Unique id of pool.
   function withdrawDelegatorRewards(bytes32 poolId)
       external
   {
       address member = msg.sender;

       _withdrawAndSyncDelegatorRewards(
           poolId,
           member
       );

       // Update stored balance with synchronized version; this prevents
   ```

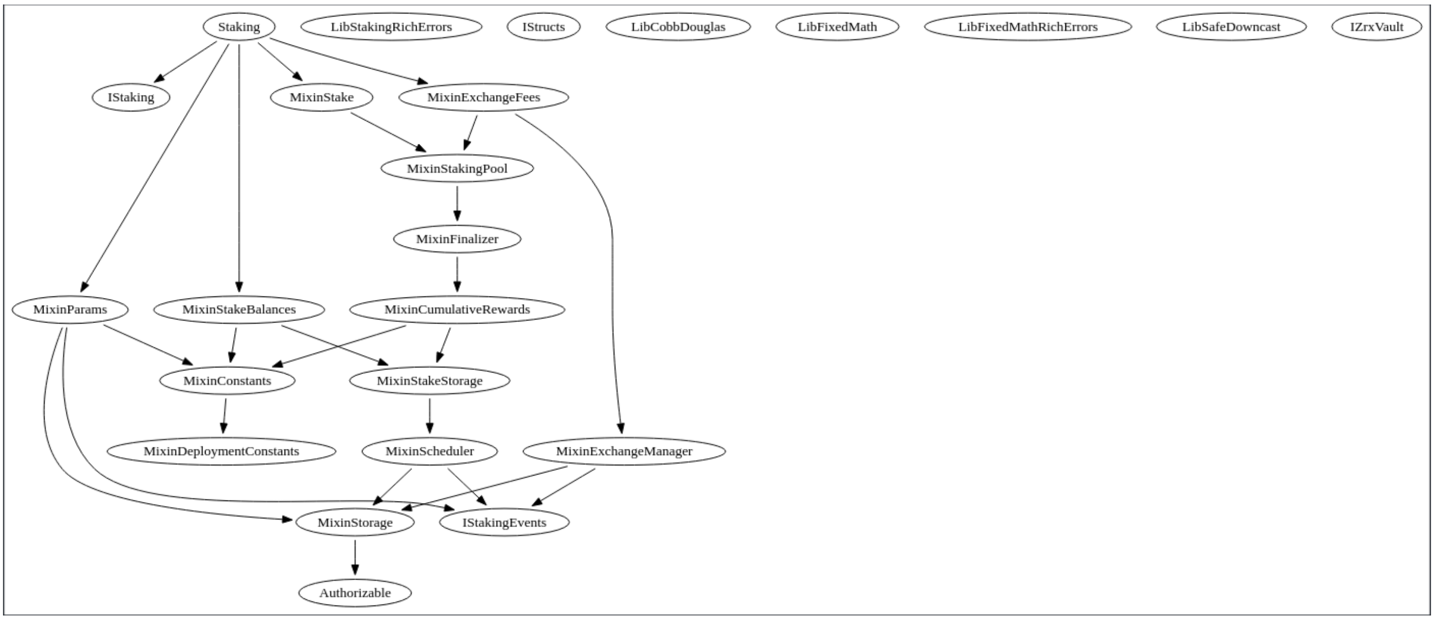
```
    // redundant withdrawals.
    _delegatedStakeToPoolByOwner[member][poolId] =
        _loadSyncedBalance(_delegatedStakeToPoolByOwner[member][poolId]);
}
```

2. Move the rest of the `MixinStakingPoolRewards` functions into `MixinFinalizer` . This change allows the `MixinStakingPoolRewards` and `MixinAbstract` files to be removed. `MixinStakingPool` can now inherit directly from `MixinFinalizer` .

After implementing all recommendations mentioned here, the inheritance graph of the staking contracts is much simpler. The previous graph is pictured here:



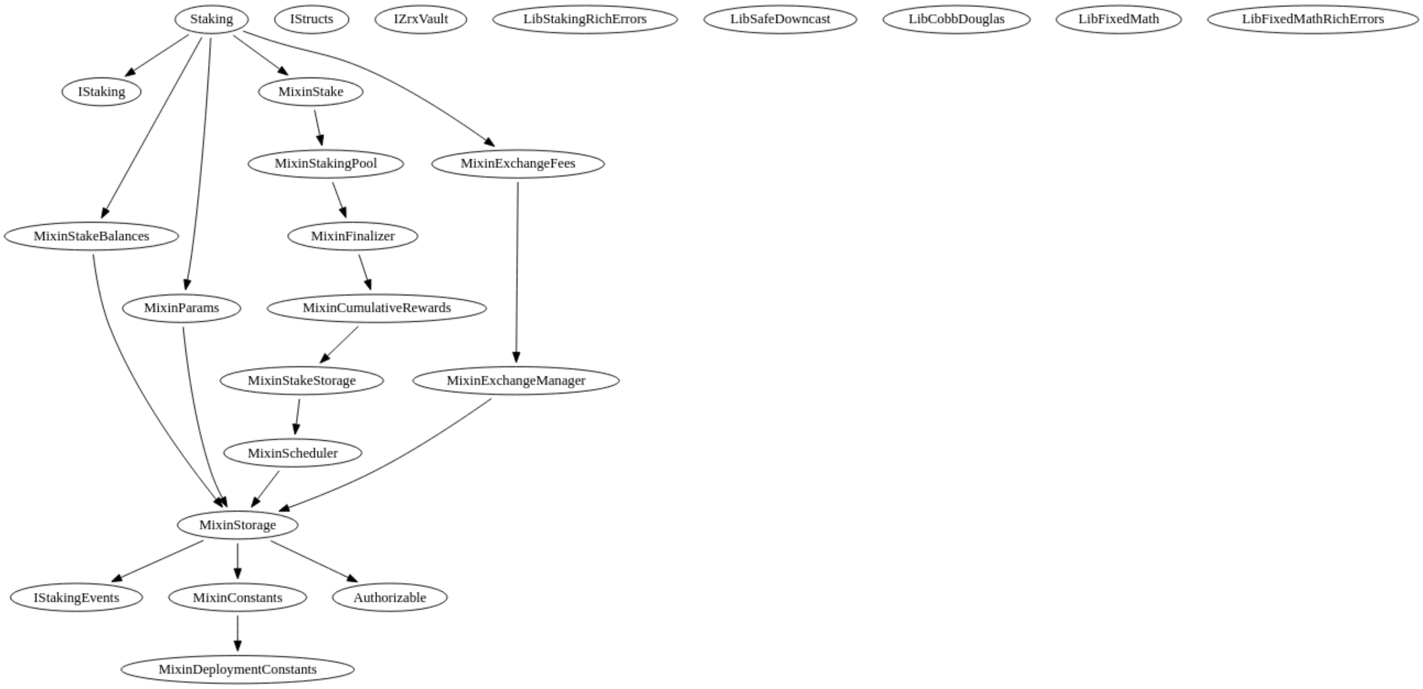
The new graph is pictured here:



Further improvements may consider:

1. Having `MixinStorage` inherit `MixinConstants` and `IStakingEvents`
2. Moving `_loadCurrentBalance` into `MixinStorage`. Currently `MixinStakeBalances` only inherits from `MixinStakeStorage` because of this function.
3. After implementing the above, `MixinExchangeFees` is no longer dependent on `MixinStakingPool` and can inherit directly from `MixinExchangeManager`

A sample inheritance graph including the above is pictured below:



5.9 Recommendation: remove confusing access to `activePoolsThisEpoch` Minor ✓ Fixed

Resolution

This is fixed in 0xProject/0x-monorepo#2276. Along with other state cleanup, these functions and `epoch % 2` indexing altogether were removed.

Description

`MixinFinalizer` provides two functions to access `activePoolsThisEpoch`:

1. `_getActivePoolsFromEpoch` returns a `storage` pointer to the mapping:

   code/contracts/staking/contracts/src/sys/MixinFinalizer.sol:L211-L225

```
/// @dev Get a mapping of active pools from an epoch.
///      This uses the formula `epoch % 2` as the epoch index in order
///      to reuse state, because we only need to remember, at most, two
///      epochs at once.
/// @return activePools The pools that were active in `epoch`.
function _getActivePoolsFromEpoch(
    uint256 epoch
)
    internal
    view
    returns (mapping (bytes32 => IStructs.ActivePool) storage activePools)
{
    activePools = _activePoolsByEpoch[epoch % 2];
    return activePools;
}
```

2. `_getActivePoolFromEpoch` invokes `_getActivePoolsFromEpoch`, then loads an `ActivePool` struct from a passed-in `poolId`:

   code/contracts/staking/contracts/src/sys/MixinFinalizer.sol:L195-L209

```
/// @dev Get an active pool from an epoch by its ID.
/// @param epoch The epoch the pool was/will be active in.
/// @param poolId The ID of the pool.
/// @return pool The pool with ID `poolId` that was active in `epoch`.
function _getActivePoolFromEpoch(
    uint256 epoch,
    bytes32 poolId
)
    internal
    view
    returns (IStructs.ActivePool memory pool)
{
    pool = _getActivePoolsFromEpoch(epoch)[poolId];
    return pool;
}
```

Ultimately, the two functions are syntax sugar for `activePoolsThisEpoch[epoch % 2]`, with the latter also accessing a value within the mapping. Because of the naming similarity, and because one calls the other, this abstraction is more confusing that simply accessing the state variable directly.

Additionally, by removing these functions and adopting the long-form syntax, `MixinExchangeFees` no longer needs to inherit `MixinFinalizer`.

5.10 Recommendation: remove `MixinFinalizer._getUnfinalizedPoolRewardsFromState` Minor Won't Fix

Resolution

The development team decided to keep this function for its optimization on storage loads. It's will still be used internally by getters that are important for client-side code.

Description

`MixinFinalizer._getUnfinalizedPoolRewardsFromState` is a simple wrapper around the library function `LibCobbDouglas.cobbDouglas`:

code/contracts/staking/contracts/src/sys/MixinFinalizer.sol:L250-L286

```
/// @dev Computes the reward owed to a pool during finalization.
/// @param pool The active pool.
/// @param state The current state of finalization.
/// @return rewards Unfinalized rewards for this pool.
function _getUnfinalizedPoolRewardsFromState(
    IStructs.ActivePool memory pool,
    IStructs.UnfinalizedState memory state
)
    private
    view
    returns (uint256 rewards)
{
    // There can't be any rewards if the pool was active or if it has
    // no stake.
    if (pool.feesCollected == 0) {
        return rewards;
    }

    // Use the cobb-douglas function to compute the total reward.
    rewards = LibCobbDouglas.cobbDouglas(
        state.rewardsAvailable,
        pool.feesCollected,
        state.totalFeesCollected,
        pool.weightedStake,
        state.totalWeightedStake,
        cobbDouglasAlphaNumerator,
        cobbDouglasAlphaDenominator
    );

    // Clip the reward to always be under
    // `rewardsAvailable - totalRewardsPaid`,
    // in case cobb-douglas overflows, which should be unlikely.
    uint256 rewardsRemaining = state.rewardsAvailable.safeSub(state.totalRewar
    if (rewardsRemaining < rewards) {
        rewards = rewardsRemaining;
    }
}
```

After implementing issue 5.11, this function is only called a single time, in `MixinFinalizer.finalizePool`:

code/contracts/staking/contracts/src/sys/MixinFinalizer.sol:L119-L129

```
// Noop if the pool was not active or already finalized (has no fees).
if (pool.feesCollected == 0) {
    return;
}

// Clear the pool state so we don't finalize it again, and to recoup
// some gas.
delete _getActivePoolsFromEpoch(prevEpoch)[poolId];

// Compute the rewards.
uint256 rewards = _getUnfinalizedPoolRewardsFromState(pool, state);
```

Because it is only used a single time, and because it obfuscates an essential library call during the finalization process, the function should be removed and folded into `finalizePool`. Additionally, the first check for `pool.feesCollected == 0` can be removed, as this case is covered in `finalizePool` already (see above).

5.11 Recommendation: remove complicating getters from `MixinStakingPoolRewards` Minor Won't Fix

Resolution

These getters are useful for client-side code, such as the staking interface.

Description

`MixinStakingPoolRewards` has two `external view` functions that contribute complexity to essential functions, as well as the overall inheritance tree:

1. `computeRewardBalanceOfOperator`, used to compute the reward balance of a pool's operator on an unfinalized pool:

   code/contracts/staking/contracts/src/staking_pools/MixinStakingPoolRewards.sol:L55-L69

   ```
   /// @dev Computes the reward balance in ETH of the operator of a pool.
   /// @param poolId Unique id of pool.
   /// @return totalReward Balance in ETH.
   function computeRewardBalanceOfOperator(bytes32 poolId)
       external
       view
       returns (uint256 reward)
   {
   ```

```
        // Because operator rewards are immediately withdrawn as WETH
        // on finalization, the only factor in this function are unfinalized
        // rewards.
        IStructs.Pool memory pool = _poolById[poolId];
        // Get any unfinalized rewards.
        (uint256 unfinalizedTotalRewards, uint256 unfinalizedMembersStake) =
            _getUnfinalizedPoolRewards(poolId);
```

2. `computeRewardBalanceOfDelegator`, used to compute the reward balance of a delegator for an unfinalized pool:

   code/contracts/staking/contracts/src/staking_pools/MixinStakingPoolRewards.sol:L80-L99

```
    /// @dev Computes the reward balance in ETH of a specific member of a poo
    /// @param poolId Unique id of pool.
    /// @param member The member of the pool.
    /// @return totalReward Balance in ETH.
    function computeRewardBalanceOfDelegator(bytes32 poolId, address member)
        external
        view
        returns (uint256 reward)
    {
        IStructs.Pool memory pool = _poolById[poolId];
        // Get any unfinalized rewards.
        (uint256 unfinalizedTotalRewards, uint256 unfinalizedMembersStake) =
            _getUnfinalizedPoolRewards(poolId);

        // Get the members' portion.
        (, uint256 unfinalizedMembersReward) = _computePoolRewardsSplit(
            pool.operatorShare,
            unfinalizedTotalRewards,
            unfinalizedMembersStake
        );
```

These two functions are the sole reason for the existence of `MixinFinalizer._getUnfinalizedPoolRewards`, one of the two functions in `MixinAbstract`:

code/contracts/staking/contracts/src/sys/MixinAbstract.sol:L40-L52

```
/// @dev Computes the reward owed to a pool during finalization.
///      Does nothing if the pool is already finalized.
/// @param poolId The pool's ID.
/// @return totalReward The total reward owed to a pool.
/// @return membersStake The total stake for all non-operator members in
///         this pool.
function _getUnfinalizedPoolRewards(bytes32 poolId)
    internal
    view
    returns (
        uint256 totalReward,
        uint256 membersStake
    );
```

These functions also necessitate two additional parameters in `MixinStakingPoolRewards._computeDelegatorReward`, which are used a single time to call `_computeUnfinalizedDelegatorReward`:

code/contracts/staking/contracts/src/staking_pools/MixinStakingPoolRewards.sol:L253-L259

```
// 1/3 Unfinalized rewards earned in `currentEpoch - 1`.
reward = _computeUnfinalizedDelegatorReward(
    delegatedStake,
    _currentEpoch,
    unfinalizedMembersReward,
    unfinalizedMembersStake
);
```

Note that `computeRewardBalanceOfOperator` and `computeRewardBalanceOfDelegator` contain the only calls to `_computeDelegatorReward` with nonzero values for the above parameters, `unfinalizedMembersReward` and `unfinalizedMembersStake`. For all essential functions, the call to `_computeUnfinalizedDelegatorReward` is a no-op.

By removing the functions `computeRewardBalanceOfOperator` and `computeRewardBalanceOfDelegator`, the following simplifications can be made:

- `_getUnfinalizedPoolRewards` can be removed from both `MixinAbstract` and `MixinFinalizer`
- The parameters `unfinalizedMembersReward` and `unfinalizedMembersStake` can be removed from `_computeDelegatorReward`
- The function `_computeUnfinalizedDelegatorReward` can be removed
- A branch of now-unused logic in `_computeDelegatorReward` can be removed

5.12 Recommendation: remove unneeded dependency on `MixinStakeBalances` Minor Won't Fix

Resolution

From the development team:

> *We're going to keep this abstraction to future-proof balance queries.*

Description

`MixinStakeBalances` has two functions used by inheriting contracts:

1. `getStakeDelegatedToPoolByOwner`, which provides shorthand to access `_delegatedStakeToPoolByOwner`:

   code/contracts/staking/contracts/src/stake/MixinStakeBalances.sol:L84-L95

   ```
   /// @dev Returns the stake delegated to a specific staking pool, by a give
   /// @param staker of stake.
   /// @param poolId Unique Id of pool.
   /// @return Stake delegated to pool by staker.
   function getStakeDelegatedToPoolByOwner(address staker, bytes32 poolId)
       public
       view
       returns (IStructs.StoredBalance memory balance)
   {
       balance = _loadCurrentBalance(_delegatedStakeToPoolByOwner[staker][poo
   ```

```
    return balance;
}
```

2. `getTotalStakeDelegatedToPool`, which provides shorthand to access `_delegatedStakeByPoolId`:

   code/contracts/staking/contracts/src/stake/MixinStakeBalances.sol:L97-L108

   ```
   /// @dev Returns the total stake delegated to a specific staking pool,
   ///      across all members.
   /// @param poolId Unique Id of pool.
   /// @return Total stake delegated to pool.
   function getTotalStakeDelegatedToPool(bytes32 poolId)
       public
       view
       returns (IStructs.StoredBalance memory balance)
   {
       balance = _loadCurrentBalance(_delegatedStakeByPoolId[poolId]);
       return balance;
   }
   ```

Each of these functions is used only a single time:

1. `MixinExchangeFees.payProtocolFee`:

   code/contracts/staking/contracts/src/fees/MixinExchangeFees.sol:L78

   ```
   uint256 poolStake = getTotalStakeDelegatedToPool(poolId).currentEpochBala
   ```

2. `MixinExchangeFees._computeMembersAndWeightedStake`:

   code/contracts/staking/contracts/src/fees/MixinExchangeFees.sol:L143-L146

   ```
   uint256 operatorStake = getStakeDelegatedToPoolByOwner(
       _poolById[poolId].operator,
       poolId
   ).currentEpochBalance;
   ```

By replacing these function invocations in `MixinExchangeFees` with the long-form access to each state variable, `MixinStakeBalances` will no longer need to be included in the inheritance trees for several contracts.

5.13 Misleading `MoveStake` event when moving stake from `UNDELEGATED` to `UNDELEGATED` Minor ✓ Fixed

Resolution

This is fixed in 0xProject/0x-monorepo#2280. If `amount` is `0` or the move is from `UNDELEGATED` to `UNDELEGATED`, the function performs an early return.

Description

Although moving stake between the same status (`UNDELEGATED <=> UNDELEGATED`) should be a no-op, calls to `moveStake` succeed even for invalid `amount` and nonsensical `poolId`. The resulting `MoveStake` event can log garbage, potentially confusing those observing events.

Examples

When moving between `UNDELEGATED` and `UNDELEGATED`, each check and function call results in a no-op, save the final event:

1. Neither `from` nor `to` are `StakeStatus.DELEGATED`, so these checks are passed:

   code/contracts/staking/contracts/src/stake/MixinStake.sol:L115-L129

```
if (from.status == IStructs.StakeStatus.DELEGATED) {
    _undelegateStake(
        from.poolId,
        staker,
        amount
    );
}

if (to.status == IStructs.StakeStatus.DELEGATED) {
    _delegateStake(
        to.poolId,
        staker,
        amount
    );
}
```

2. The primary state changing function, `_moveStake`, immediately returns because the `from` and `to` balance pointers are equivalent:

   code/contracts/staking/contracts/src/stake/MixinStakeStorage.sol:L47-L49

   ```
   if (_arePointersEqual(fromPtr, toPtr)) {
       return;
   }
   ```

3. Finally, the `MoveStake` event is invoked, which can log completely invalid values for `amount`, `from.poolId`, and `to.poolId`:

   code/contracts/staking/contracts/src/stake/MixinStake.sol:L141-L148

   ```
   emit MoveStake(
       staker,
       amount,
       uint8(from.status),
       from.poolId,
       uint8(to.status),
       to.poolId
   );
   ```

Recommendation

If `amount` is 0 or if moving between `UNDELEGATED` and `UNDELEGATED`, this function should no-op or revert. An explicit check for this case should be made near the start of the function.

5.14 The staking contracts contain several artifacts of a quickly-changing codebase Minor ✓ Fixed

Resolution

These issues were addressed in a variety of fixes, most notably 0xProject/0x-monorepo#2262.

Examples

1. `address payable` is used repeatedly, but payments use WETH:
   - `MixinStakingPool.createStakingPool`:

     code/contracts/staking/contracts/src/staking_pools/MixinStakingPool.sol:L54

     ```
     address payable operator = msg.sender;
     ```

   - `ZrxVault.stakingProxyAddress`:

     code/contracts/staking/contracts/src/ZrxVault.sol:L38

     ```
     address payable public stakingProxyAddress;
     ```

   - `ZrxVault.setStakingProxy`:

     code/contracts/staking/contracts/src/ZrxVault.sol:L76

     ```
     function setStakingProxy(address payable _stakingProxyAddress)
     ```

   - `IZrxVault.setStakingProxy`:

     code/contracts/staking/contracts/src/interfaces/IZrxVault.sol:L53

     ```
     function setStakingProxy(address payable _stakingProxyAddress)
     ```

   - `struct IStructs.Pool`:

code/contracts/staking/contracts/src/interfaces/IStructs.sol:L114

```
address payable operator;
```

- `MixinStake.stake`:

  code/contracts/staking/contracts/src/stake/MixinStake.sol:L38

  ```
  address payable staker = msg.sender;
  ```

- `MixinStake.unstake`:

  code/contracts/staking/contracts/src/stake/MixinStake.sol:L63

  ```
  address payable staker = msg.sender;
  ```

- `MixinStake.moveStake`:

  code/contracts/staking/contracts/src/stake/MixinStake.sol:L119

  ```
  address payable staker = msg.sender;
  ```

- `MixinStake._delegateStake`:

  code/contracts/staking/contracts/src/stake/MixinStake.sol:L181

  ```
  address payable staker,
  ```

- `MixinStake._undelegateStake`:

  code/contracts/staking/contracts/src/stake/MixinStake.sol:L210

  ```
  address payable staker,
  ```

2. Some identifiers are used multiple times for different purposes:

   - `currentEpoch` is:

        - A state variable:

          code/contracts/staking/contracts/src/immutable/MixinStorage.sol:L86

          ```
          uint256 public currentEpoch = INITIAL_EPOCH;
          ```

        - A function parameter:

          code/contracts/staking/contracts/src/staking_pools/MixinStakingPoolRewards.

          ```
          uint256 currentEpoch,
          ```

        - A struct field:

          code/contracts/staking/contracts/src/interfaces/IStructs.sol:L62

          ```
          uint32 currentEpoch;
          ```

3. Several comments are out of date:

    - Many struct comments reference fees and rewards denominated in ETH, while only WETH is used:

      code/contracts/staking/contracts/src/interfaces/IStructs.sol:L36-L38

      ```
      /// @param rewardsAvailable Rewards (ETH) available to the epoch
      ///        being finalized (the previous epoch). This is simply the ba
      ///        of the contract at the end of the epoch.
      ```

    - `UnfinalizedState.totalFeesCollected` should specify that it is tracking fees attributed to a pool. Fees not attributed to a pool are still collected, but are not recorded:

      code/contracts/staking/contracts/src/interfaces/IStructs.sol:L41

      ```
      /// @param totalFeesCollected The total fees collected for the epoch b
      ```

- `UnfinalizedState.totalWeightedStake` is copy-pasted from `totalFeesCollected`:

  code/contracts/staking/contracts/src/interfaces/IStructs.sol:L42

  ```
  /// @param totalWeightedStake The total fees collected for the epoch b
  ```

- `Pool.initialized` seems to be copy-pasted from an older version of the struct `StoredBalance` or `StakeBalance`:

  code/contracts/staking/contracts/src/interfaces/IStructs.sol:L108

  ```
  /// @param initialized True iff the balance struct is initialized.
  ```

4. The final contracts produce several compiler warnings:

   - Several functions are intentionally marked `view` to allow overriding implementations to read from state. These can be silenced by adding `block.timestamp;` or similar statements to the functions.

   - One function is erroneously marked `view`, and should be changed to pure:

     **code/contracts/staking/contracts/src/staking_pools/MixinStakingPoolRewards.sol:L
     L330**

     ```
     /// @dev Computes the unfinalized rewards earned by a delegator in the
     /// @param unsyncedStake Unsynced delegated stake to pool by staker
     /// @param currentEpoch The epoch in which this call is executing
     /// @param unfinalizedMembersReward Unfinalized total members reward (
     /// @param unfinalizedMembersStake Unfinalized total members stake (i
     /// @return reward Balance in WETH.
     function _computeUnfinalizedDelegatorReward(
         IStructs.StoredBalance memory unsyncedStake,
         uint256 currentEpoch,
         uint256 unfinalizedMembersReward,
         uint256 unfinalizedMembersStake
     )
         private
         view
     ```

```
    returns (uint256)
{
```

5.15 Remove unneeded fields from `StoredBalance` and `Pool` structs Minor ✓ Fixed

Resolution

This is fixed in 0xProject/0x-monorepo#2248. As part of a larger refactor, these fields were removed.

Description

Both structs have fields that are only written to, and never read:

1. `StoredBalance.isInitialized`:

   code/contracts/staking/contracts/src/interfaces/IStructs.sol:L61

   ```
   bool isInitialized;
   ```

2. `Pool.initialized`:

   code/contracts/staking/contracts/src/interfaces/IStructs.sol:L113

   ```
   bool initialized;
   ```

Recommendation

The unused fields should be removed.

5.16 Remove unnecessary fallback function in Staking contract Minor ✓ Fixed

Resolution

This is fixed in 0xProject/0x-monorepo#2277.

Description

The `Staking` contract has a `payable` fallback function that is never used. Because it is used with a proxy contract, this pattern introduces silent failures when calls are made to the contract with no matching function selector.

Recommendation

Remove the fallback function from `Staking`.

5.17 Pool IDs can just be incrementing integers Minor ✓ Fixed

Resolution

This is fixed in 0xProject/0x-monorepo#2250. Pool IDs now start at 1 and increment by 1 each time.

Description

Pool IDs are currently `bytes32` values that increment by `2**128`. After discussion with the development team, it seems that this was in preparation for a feature that was ultimately not used. Pool IDs should instead just be incrementing integers.

Examples

code/contracts/staking/contracts/src/immutable/MixinConstants.sol:L30-L34

```
// The upper 16 bytes represent the pool id, so this would be pool id 1. See
bytes32 constant internal INITIAL_POOL_ID = 0x00000000000000000000000000000001

// The upper 16 bytes represent the pool id, so this would be an increment of
uint256 constant internal POOL_ID_INCREMENT_AMOUNT = 0x00000000000000000000000
```

code/contracts/staking/contracts/src/staking_pools/MixinStakingPool.sol:L271-L280

```
/// @dev Computes the unique id that comes after the input pool id.
/// @param poolId Unique id of pool.
/// @return Next pool id after input pool.
function _computeNextStakingPoolId(bytes32 poolId)
    internal
    pure
    returns (bytes32)
{
    return bytes32(uint256(poolId).safeAdd(POOL_ID_INCREMENT_AMOUNT));
}
```

Recommendation

Make pool IDs `uint256` values and simply add 1 to generate the next ID.

5.18 `LibProxy.proxyCall()` may overwrite important memory Minor ✓ Fixed

Resolution

This is fixed in 0xProject/0x-monorepo#2301. This function has been rewritten in Solidity and now avoids manual memory management.

Description

`LibProxy.proxyCall()` copies from call data to memory, starting at address 0:

code/contracts/staking/contracts/src/libs/LibProxy.sol:L52-L71

```
assembly {
    // store selector of destination function
    let freeMemPtr := 0
    if gt(customEgressSelector, 0) {
        mstore(0x0, customEgressSelector)
        freeMemPtr := add(freeMemPtr, 4)
    }
    // adjust the calldata offset, if we should ignore the selector
    let calldataOffset := 0
    if gt(ignoreIngressSelector, 0) {
        calldataOffset := 4
    }

    // copy calldata to memory
    calldatacopy(
        freeMemPtr,
        calldataOffset,
        calldatasize()
    )
```

The first 64 bytes of memory are treated as “scratch space” by the Solidity compiler. Writing beyond that point is dangerous, as it will overwrite the free memory pointer and the “zero slot” which is where length-0 arrays point.

Although the current callers of `proxyCall()` don’t appear to use any memory after calling `proxyCall()`, future changes to the code may introduce very serious and subtle bugs due to this unsafe handling of memory.

Recommendation

Use the actual free memory pointer to determine where it’s safe to write to memory.

6 Tool-Based Analysis

Several tools were used to perform automated analysis of the reviewed contracts. These issues were reviewed by the audit team, and relevant issues are listed in the Issue Details section.

6.1 MythX

MythX is a security analysis API for Ethereum smart contracts. It performs multiple types of analysis, including fuzzing and symbolic execution, to detect many common vulnerability types. The tool was used for automated vulnerability discovery for all audited contracts and libraries. More details on MythX can be found at mythx.io.

The full set of MythX results for both the exchange and staking contracts are available in a separate report.

6.2 Surya

Surya is an utility tool for smart contract systems. It provides a number of visual outputs and information about structure of smart contracts. It also supports querying the function call graph in multiple ways to aid in the manual inspection and control flow analysis of contracts.

Below is a complete list of functions with their visibility and modifiers:

Sūrya's Description Report

Files Description Table

File Name	SHA-1 Hash
ReadOnlyProxy.sol	6ec64526446ebff87ec5528ee3b2786338cc4fa0
Staking.sol	67ddcb9ab75e433882e28d9186815990b7084c61
StakingProxy.sol	248f562d014d0b1ca6de3212966af3e52a7deef1
ZrxVault.sol	6c3249314868a2f5d0984122e8ab1413a5b521c9
fees/MixinExchangeFees.sol	9ac3b696baa8ba09305cfc83d3c08f17d9d528e1
fees/MixinExchangeManager.sol	46f48136a49919cdb5588dc1b3d64c977c3367f2
immutable/MixinConstants.sol	97c2ac83ef97a09cfd485cb0d4b119ba0902cc79
immutable/MixinDeploymentConstants.sol	424f22c45df8e494c4a78f239ea07ff0400d694b
immutable/MixinStorage.sol	8ad475b0e424e7a3ff65eedf2e999cba98f414c8
interfaces/IStaking.sol	ec1d7f214e3fd40e14716de412deee9769359bc0
interfaces/IStakingEvents.sol	25f16b814c4df9d2002316831c3f727d858456c4
interfaces/IStakingProxy.sol	02e35c6b51e08235b2a01d30a8082d60d9d61bee
interfaces/IStorage.sol	eeaa798c262b46d1874e904cf7de0423d4132cee
interfaces/IStorageInit.sol	b9899b03e474ea5adc3b4818a4357f71b8d288d4
interfaces/IStructs.sol	fee17d036883d641afb1222b75eec8427f3cdb96

File Name	SHA-1 Hash
interfaces/IZrxVault.sol	9067154651675317e000cfa92de9741e50c1c809
libs/LibCobbDouglas.sol	242d62d71cf8bc09177d240c0db59b83f9bb4e96
libs/LibFixedMath.sol	36311e7be09a947fa4e6cd8c544cacd13d65833c
libs/LibFixedMathRichErrors.sol	39cb3e07bbce3272bbf090e87002d5834d288ec2
libs/LibProxy.sol	29abe52857a782c8da39b053cc54e02e295c1ae2
libs/LibSafeDowncast.sol	ae16ed2573d64802793320253b060b9507729c3d
libs/LibStakingRichErrors.sol	f5868ef6066a18277c932e59c0a516ec58920b00
stake/MixinStake.sol	ade59ed356fe72521ffd2ef12ff8896c852f11f8
stake/MixinStakeBalances.sol	cde6ca1a6200570ba18dd6d392ffabf68c2bb464
stake/MixinStakeStorage.sol	cadf34d9d341efd2a85dd13ec3cd4ce8383e0f73
staking_pools/MixinCumulativeRewards.sol	664ea3e35376c81492457dc17832a4d0d602c8ae
staking_pools/MixinStakingPool.sol	74ba9cb2db29b8dd6376d112e9452d117a391b18
staking_pools/MixinStakingPoolRewards.sol	a3b4e5c9b1c3568c94923e2dd9a93090ebdf8536
sys/MixinAbstract.sol	99fd4870c20d8fa03cfa30e8055d3dfb348ed5cd
sys/MixinFinalizer.sol	cc658ed07241c1804cec75b12203be3cd8657b9b
sys/MixinParams.sol	7b395f4da7ed787d7aa4eb915f15377725ff8168
sys/MixinScheduler.sol	2fab6b83a6f9e1d0dd1b1bdcea4b129d166aef1d

Contracts Description Table

Contract	Type	Bases
L	**Function Name**	**Visibility**
ReadOnlyProxy	Implementation	MixinStorage
L	<Fallback>	External ❗
L	revertDelegateCall	External ❗
Staking	Implementation	IStaking, MixinPa MixinStake, MixinExchangeF

Contract	Type	Bases
└	<Fallback>	External ❗
└	init	Public ❗
StakingProxy	Implementation	IStakingProxy, Mixin
└	<Constructor>	Public ❗
└	<Fallback>	External ❗
└	attachStakingContract	External ❗
└	detachStakingContract	External ❗
└	setReadOnlyMode	External ❗
└	batchExecute	External ❗
└	_assertValidStorageParams	Internal 🔒
└	_attachStakingContract	Internal 🔒
ZrxVault	Implementation	Authorizable, IZrx
└	<Constructor>	Public ❗
└	setStakingProxy	External ❗
└	enterCatastrophicFailure	External ❗
└	setZrxProxy	External ❗
└	depositFrom	External ❗
└	withdrawFrom	External ❗
└	withdrawAllFrom	External ❗
└	balanceOf	External ❗
└	_withdrawFrom	Internal 🔒
└	_assertSenderIsStakingProxy	Private 🔐
└	_assertInCatastrophicFailure	Private 🔐
└	_assertNotInCatastrophicFailure	Private 🔐

Contract	Type	Bases
MixinExchangeFees	Implementation	MixinExchangeMa MixinStakingPo MixinFinalize
└	payProtocolFee	External ❗
└	getActiveStakingPoolThisEpoch	External ❗
└	_computeMembersAndWeightedStake	Private 🔐
└	_assertValidProtocolFee	Private 🔐
MixinExchangeManager	Implementation	IStakingEvent MixinStorage
└	addExchangeAddress	External ❗
└	removeExchangeAddress	External ❗
MixinConstants	Implementation	MixinDeploymentCo
MixinDeploymentConstants	Implementation	
└	getWethContract	Public ❗
└	getZrxVault	Public ❗
MixinStorage	Implementation	MixinConstant Authorizable
IStaking	Interface	
└	moveStake	External ❗
└	payProtocolFee	External ❗
└	stake	External ❗
IStakingEvents	Interface	
IStakingProxy	Interface	
└	<Fallback>	External ❗
└	attachStakingContract	External ❗
└	detachStakingContract	External ❗

Contract	Type	Bases
IStorage	Interface	
└	stakingContract	External ❗
└	readOnlyProxy	External ❗
└	readOnlyProxyCallee	External ❗
└	nextPoolId	External ❗
└	numMakersByPoolId	External ❗
└	currentEpoch	External ❗
└	currentEpochStartTimeInSeconds	External ❗
└	protocolFeesThisEpochByPool	External ❗
└	activePoolsThisEpoch	External ❗
└	validExchanges	External ❗
└	epochDurationInSeconds	External ❗
└	rewardDelegatedStakeWeight	External ❗
└	minimumPoolStake	External ❗
└	maximumMakersInPool	External ❗
└	cobbDouglasAlphaNumerator	External ❗
└	cobbDouglasAlphaDenominator	External ❗
IStorageInit	Interface	
└	init	External ❗
IStructs	Interface	
IZrxVault	Interface	
└	setStakingProxy	External ❗
└	enterCatastrophicFailure	External ❗
└	setZrxProxy	External ❗
└	depositFrom	External ❗

Contract	Type	Bases
L	withdrawFrom	External ❗
L	withdrawAllFrom	External ❗
L	balanceOf	External ❗
LibCobbDouglas	Library	
L	cobbDouglas	Internal 🔒
LibFixedMath	Library	
L	one	Internal 🔒
L	add	Internal 🔒
L	sub	Internal 🔒
L	mul	Internal 🔒
L	div	Internal 🔒
L	mulDiv	Internal 🔒
L	uintMul	Internal 🔒
L	abs	Internal 🔒
L	invert	Internal 🔒
L	toFixed	Internal 🔒
L	toFixed	Internal 🔒
L	toFixed	Internal 🔒
L	toFixed	Internal 🔒
L	toInteger	Internal 🔒
L	ln	Internal 🔒
L	exp	Internal 🔒
L	_mul	Private 🔐
L	_div	Private 🔐
L	_add	Private 🔐
LibFixedMathRichErrors	Library	

Contract	Type	Bases
L	SignedValueError	Internal 🔒
L	UnsignedValueError	Internal 🔒
L	BinOpError	Internal 🔒
LibProxy	Library	
L	proxyCall	Internal 🔒
LibSafeDowncast	Library	
L	downcastToUint96	Internal 🔒
L	downcastToUint64	Internal 🔒
L	downcastToUint32	Internal 🔒
LibStakingRichErrors	Library	
L	OnlyCallableByExchangeError	Internal 🔒
L	ExchangeManagerError	Internal 🔒
L	InsufficientBalanceError	Internal 🔒
L	OnlyCallableByPoolOperatorOrMakerError	Internal 🔒
L	MakerPoolAssignmentError	Internal 🔒
L	BlockTimestampTooLowError	Internal 🔒
L	OnlyCallableByStakingContractError	Internal 🔒
L	OnlyCallableIfInCatastrophicFailureError	Internal 🔒
L	OnlyCallableIfNotInCatastrophicFailureError	Internal 🔒
L	OperatorShareError	Internal 🔒
L	PoolExistenceError	Internal 🔒
L	InvalidProtocolFeePaymentError	Internal 🔒
L	InvalidStakeStatusError	Internal 🔒
L	InitializationError	Internal 🔒
L	InvalidParamValueError	Internal 🔒
L	ProxyDestinationCannotBeNilError	Internal 🔒

Contract	Type	Bases
L	PreviousEpochNotFinalizedError	Internal 🔒
MixinStake	Implementation	MixinStakingP
L	stake	External ❗
L	unstake	External ❗
L	moveStake	External ❗
L	_delegateStake	Private 🔐
L	_undelegateStake	Private 🔐
L	_getBalancePtrFromStatus	Private 🔐
MixinStakeBalances	Implementation	MixinStakeStor
L	getGlobalActiveStake	External ❗
L	getGlobalInactiveStake	External ❗
L	getGlobalDelegatedStake	External ❗
L	getTotalStake	External ❗
L	getActiveStake	External ❗
L	getInactiveStake	External ❗
L	getStakeDelegatedByOwner	External ❗
L	getWithdrawableStake	Public ❗
L	getStakeDelegatedToPoolByOwner	Public ❗
L	getTotalStakeDelegatedToPool	Public ❗
L	_computeWithdrawableStake	Internal 🔒
MixinStakeStorage	Implementation	MixinSchedul
L	_moveStake	Internal 🔒
L	_loadSyncedBalance	Internal 🔒
L	_loadUnsyncedBalance	Internal 🔒
L	_increaseCurrentAndNextBalance	Internal 🔒
L	_decreaseCurrentAndNextBalance	Internal 🔒

Contract	Type	Bases
L	_increaseNextBalance	Internal 🔒
L	_decreaseNextBalance	Internal 🔒
L	_storeBalance	Private 🔐
L	_arePointersEqual	Private 🔐
MixinCumulativeRewards	Implementation	MixinStakeBalar
L	_initializeCumulativeRewards	Internal 🔒
L	_isCumulativeRewardSet	Internal 🔒
L	_forceSetCumulativeReward	Internal 🔒
L	_computeMemberRewardOverInterval	Internal 🔒
L	_getMostRecentCumulativeReward	Internal 🔒
L	_getCumulativeRewardAtEpoch	Internal 🔒
MixinStakingPool	Implementation	MixinAbstrac MixinStakingPoolR
L	createStakingPool	External ❗
L	decreaseStakingPoolOperatorShare	External ❗
L	joinStakingPoolAsMaker	External ❗
L	addMakerToStakingPool	External ❗
L	removeMakerFromStakingPool	External ❗
L	getStakingPoolIdOfMaker	Public ❗
L	getStakingPool	Public ❗
L	_addMakerToStakingPool	Internal 🔒
L	_computeNextStakingPoolId	Internal 🔒
L	_assertStakingPoolExists	Internal 🔒
L	_assertNewOperatorShare	Private 🔐
L	_assertSenderIsPoolOperatorOrMaker	Private 🔐

Contract	Type	Bases
MixinStakingPoolRewards	Implementation	MixinAbstrac MixinCumulativeRe
└	withdrawDelegatorRewards	External ❗
└	computeRewardBalanceOfOperator	External ❗
└	computeRewardBalanceOfDelegator	External ❗
└	_withdrawAndSyncDelegatorRewards	Internal 🔒
└	_syncPoolRewards	Internal 🔒
└	_computePoolRewardsSplit	Internal 🔒
└	_computeDelegatorReward	Private 🔐
└	_computeUnfinalizedDelegatorReward	Private 🔐
└	_increasePoolRewards	Private 🔐
└	_decreasePoolRewards	Private 🔐
MixinAbstract	Implementation	
└	finalizePool	Public ❗
└	_getUnfinalizedPoolRewards	Internal 🔒
MixinFinalizer	Implementation	MixinStakingPoolR
└	endEpoch	External ❗
└	finalizePool	Public ❗
└	_getUnfinalizedPoolRewards	Internal 🔒
└	_getActivePoolFromEpoch	Internal 🔒
└	_getActivePoolsFromEpoch	Internal 🔒
└	_wrapEth	Internal 🔒
└	_getAvailableWethBalance	Internal 🔒
└	_getUnfinalizedPoolRewardsFromState	Private 🔐
└	_creditRewardsToPool	Private 🔐

Contract	Type	Bases
MixinParams	Implementation	IStakingEvent MixinStorage
L	setParams	External ❗
L	getParams	External ❗
L	_initMixinParams	Internal 🔒
L	_assertParamsNotInitialized	Internal 🔒
L	_setParams	Private 🔐
MixinScheduler	Implementation	IStakingEvent MixinStorage
L	getCurrentEpochEarliestEndTimeInSeconds	Public ❗
L	_initMixinScheduler	Internal 🔒
L	_goToNextEpoch	Internal 🔒
L	_assertSchedulerNotInitialized	Internal 🔒

Legend

Symbol	Meaning
🛑	Function can modify state
💵	Function is payable

Appendix 1 - Disclosure

ConsenSys Diligence ("CD") typically receives compensation from one or more clients (the "Clients") for performing the analysis contained in these reports (the "Reports"). The Reports may be distributed through other means, including via ConsenSys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of Solidity code and only the Solidity code we note as being within the scope of our review within this report. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond Solidity that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) – on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

LINKS TO OTHER WEB SITES FROM THIS WEB SITE You may, through hypertext or other computer links, gain access to web sites operated by persons other than ConsenSys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites’ owners. You agree that ConsenSys and CD are not responsible for the content or operation of such Web sites, and that ConsenSys and CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that ConsenSys and CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. ConsenSys and CD assumes no responsibility for the use of third party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

TIMELINESS OF CONTENT The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice. Unless indicated otherwise, by ConsenSys and CD.