FATHOM DAO.
REAUDIT

0XORIO

MAY 10, 2023

CONTENTS

1 INTRO

1.1 DISCLAIMER

The audit makes no statements or warranties about the utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about the fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

1.2 ABOUT OXORIO

Oxorio is a young but rapidly growing audit and consulting company in the field of the blockchain industry, providing consulting and security audits for organizations from all over the world. Oxorio has participated in multiple blockchain projects during which smart contract systems were designed and deployed by the company.

Oxorio is the creator, maintainer, and major contributor of several blockchain projects and employs more than 5 blockchain specialists to analyze and develop smart contracts.

Our contacts:

- oxor.io
- ping@oxor.io
- Github
- Linkedin
- Twitter

1.3 SECURITY ASSESSMENT METHODOLOGY

A group of auditors is involved in the work on this audit. Each of them checks the provided source code independently of each other in accordance with the security assessment methodology described below:

1. Project architecture review

Study the source code manually to find errors and bugs.

2. Check the code for known vulnerabilities from the list

Conduct a verification process of the code against the constantly updated list of already known vulnerabilities maintained by the company.

3. Architecture and structure check of the security model

Study the project documentation and its comparison against the code including the study of the comments and other technical papers.

4. Result's cross-check by different auditors

Normally the research of the project is done by more than two auditors. This is followed by a step of mutual cross-check process of the audit results between different task performers.

5. Report consolidation

Consolidation of the audited report from multiple auditors.

6. Reaudit of new editions

After the provided review and fixes from the client, the found issues are being double-checked. The results are provided in the new version of the audit.

7. Final audit report publication

The final audit version is provided to the client and also published on the official website of the company.

1.4 FINDINGS CLASSIFICATION

1.4.1 Severity Level Reference

The following severity levels were assigned to the issues described in the report:

- **CRITICAL**: A bug leading to assets theft, locked fund access, or any other loss of funds due to transfer to unauthorized parties.
- **MAJOR**: A bug that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
- **WARNING**: A bug that can break the intended contract logic or expose it to DDoS attacks.
- **INFO**: Minor issue or recommendation reported to / acknowledged by the client's team.

1.4.2 Status Level Reference

Based on the feedback received from the client's team regarding the list of findings discovered by the contractor, the following statuses were assigned to the findings:

- **NEW**: Waiting for the project team's feedback.
- **FIXED**: Recommended fixes have been applied to the project code and the identified issue no longer affects the project's security.
- **ACKNOWLEDGED**: The project team is aware of this finding. Recommended fixes for this finding are planned to be made. This finding does not affect the overall security of the project.
- **NO ISSUE**: Finding does not affect the overall security of the project and does not violate the logic of its work.
- **DISMISSED**: The issue or recommendation was dismissed by the client.

1.5 PROJECT OVERVIEW

Fathom is a decentralized, community governed protocol. Locking FTHM tokens in DAO vault will allow you to put forward proposals and vote on them.

1.6 AUDIT SCOPE

The scope of the audit includes the following smart contracts at:

- Treasury contracts
- Governance contracts
- DAO Tokens contracts
- Staking contracts

The audited commit identifier is `5e9f3a23bd2b6deb9babe1a3ad984fd84cf51b7a`

The reaudited commit identifier is `30aa0beb27eb21ad1fef4675a7ef9f1ee01f61a5`

2 FINDINGS REPORT

OXORIO

2.1 CRITICAL

2.1.1 There's no `owners` array length validation in the constructor of `MultiSigWallet`

SEVERITY	CRITICAL
STATUS	FIXED

Description

In the MultiSigWallet`s constructor there's no checking that the number of `owners` is less than or equal `MAX_OWNER_COUNT`. If the contract is created with `owners` with length more than `MAX_OWNER_COUNT` then that makes calls to `addOwner`, `changeRequirement` and `removeOwner` (which uses call `changeRequirement`) functions impossible because they use modifier `validRequirement` with this `require` statement:

```
require(ownerCount <= MAX_OWNER_COUNT && _required <= ownerCount && _required != 0 &&
ownerCount != 0, "MultiSig: Invalid requirement");
_;
```

Recommendation

We recommend adding `owners` array length validation to `MultiSigWallet` constructor:

```
require(_owners.length <= MAX_OWNER_COUNT, "owners limit reached");
```

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The implementation of the recommendation has led to new problems.

In `MultiSigWallet` contract `constructor` misses `OwnerAddition` event. If external services or backend monitoring is used, `_owners` added with `constructor` will not be included in the statistics.

We recommend adding the following line to `constructor`:

```
emit OwnerAddition(owner);
```

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in `MultiSigWallet#L117`.

2.1.2 Adding a new owner doesn't change necessary amount of signatures in `MultiSigWallet`

SEVERITY	CRITICAL
STATUS	FIXED

Description

In the function `addOwner` the owner is added without changing the parameter `numConfirmationsRequired`. In a situation, for example, where signatures of 2 out of 4 owners are required, it results in that when the owner is added, there will be 2 out of 5, and it requires less than a half of the signatures to manage the functions of the contract, so the contract could be compromised.

Recommendation

We recommend adding this call into function `addOwner`:

```
changeRequirement(numConfirmationsRequired+1);
```

Update

Fathom's response

Implemented Auditors Recommendation. with slight change:

```
changeRequirement(numConfirmationsRequired + _owners.length);
```

2.1.3 Removing owner without `revokeConfirmation` transaction in `MultiSigWallet`

SEVERITY	CRITICAL
STATUS	FIXED

Description

In the function `removeOwner` the owner is being removed without revocation of transaction signatures, where they've signed. This creates a situation where the signatures of non-existent owners may be used. For example, like in the following scenario:

1. There are signatures of 3 out of 5 owners.
2. 3 owners opposed the signing of the transaction, and 2 owners approved it.
3. 3 owners called `removeOwner` for 2 owners, who previously signed the transaction.
4. Then, one of the 3 remaining owners , using signatures of non-existent owners are able to execute the transaction.

Recommendation

We recommend adding signature revocation mechanisms for signatures of the removed owners to the function `removeOwner`.

Update

Fathom response

Implemented Auditors Recommendation.

Oxorio's response

This logic disables all transactions up to the current moment.

```
modifier notDisabled(uint _txIndex) {
    require(_txIndex >= lastDisabledTransactionIndex, "MultiSig: old txs has been disabled");
    _;
}
```

This allows to manipulate with transaction acceptance, for example, it is possible to execute a transaction that removes a user before executing a transaction that collects confirmations. Thus, the transaction that has collected confirmations will be disabled and

will not be able to be executed.
We propose refactoring this code according to the recommendation.

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in `MultiSigWallet#L186-L194`, but it led to new major vulnerabilities.

The `removeOwner` uses a loop to remove the owner, but if the owner is malicious, it can provide enough proposals and confirm them itself, so that removing it by using the loop would simply be impossible, as it would consume too much gas.

And also the contract will forever lose the option to remove the owner if the owner has more than 1 confirmed transaction, because it first takes the total length of the array of confirmed transactions of the owner, but the length of the array in the loop is reduced after each removal and the index of the transaction to be removed only increases, resulting in an attempt to access the element out of bounds and failure to remove the owner.

We recommend changing the owner removal logic.

Fathom's response

The fix was implemented in commit with identifier 4175625c23eeb27907a8dc5a5e9dd40c7593c7b6.
We accept that all the previous transactions that were confirmed and submitted will be lost, and we will only submit the transactions and confirm the transactions that is really needed again. It helps in filtering out the unnecessary transactions as well.

2.1.4 There is no function that implements the `_cancel` proposal in `MainTokenGovernor`

SEVERITY	CRITICAL
STATUS	FIXED

Description

The contract `MainTokenGovernor` lacks a function that would implement the internal function `_cancel`, that allows you to cancel the execution of `proposal` with

`TimelockController`. This can make it impossible to cancel the execution of a potentially dangerous call.

Recommendation

We recommend adding logic that would allow you to cancel the execution of `proposal` and call the internal function `_cancel`.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in commit with identifier 2c40cfad6436ed2a9d9563213e4db222aae31f5e.

2.1.5 Changing the `timelock` address may cause re-execution of the proposals in `GovernorTimelockControl`

SEVERITY	**CRITICAL**
STATUS	**FIXED**

Description

A change of the `timelock` parameter in the `GovernorTimelockControl` contract can lead to already executed `proposals` being able to be executed again. This is connected to the fact that the execution status of the transaction is saved only in the `TimelockController` contract, and the `GovernorTimelockControl` contract makes calls to the `TimelockController` functions to get the `proposals` status in the `state` function.

Recommendation

We recommend adding a separate mapping to the `GovernorTimelockControl` contract that would save information about the status of `proposal` and functions that would allow to update that status.

Update

Fathom's response

Implemented Auditors Recommendation.
Added:

```
mapping(uint256 => bool) private isProposalExecuted;
```

Oxorio's response

The recommendation has not been fully implemented.

We recommend changing the work with the `state` function to:
`isProposalExecuted[proposalId] == true`

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in `GovernorTimelockControl#L61`.

2.1.6 The `initVault` and `initAdminAndOperator` functions can be initialized from any address in the `VaultPackage` contract

SEVERITY	CRITICAL
STATUS	FIXED

Description

In the `VaultPackage` contract the `initVault` and `initAdminAndOperator` functions can be called from any address. This could result in a potential attacker being able to intercept control for both `initVault` and `initAdminAndOperator` calls.

Recommendation

We suggest two solutions to this problem:

- Combine the `initVault` and `initAdminAndOperator` functions into one `initialize` function and pass `calldata` to the VaultProxy constructor in the `_data` parameter.
- Make a call to the `initVault` function on behalf of the `DEFAULT_ADMIN_ROLE`, and pass the `initVault` parameters just as `calldata` in the VaultProxy constructor.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in commit with identifier 2c40cfad6436ed2a9d9563213e4db222aae31f5e.

2.1.7 There is no check that `stream` is active in the `StakingHandler` contract

SEVERITY	CRITICAL
STATUS	FIXED

Description

In the `StakingHandler` contract the `withdrawAllStreams` and `withdrawStream` functions do not have a check that `stream` is active. In the case of `withdrawAllStreams` this causes the function to use the entire `streams` array each time with active and inactive `streams` and, if there are not enough tokens on `VaultPackage`, the entire transaction will be `reverted`. In the case of `withdrawStream`, this can lead to `reverted` transaction, or unauthorized withdrawal of tokens from `VaultPackage`.

Recommendation

We recommend adding to the `withdrawAllStreams` and `withdrawStream` functions a check that the output from `stream` has the status `ACTIVE`.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in commit with identifier 2c40cfad6436ed2a9d9563213e4db222aae31f5e.

2.1.8 Calling the `updateConfig` function may block the work of the `StakingHandlers` contract

SEVERITY	CRITICAL
STATUS	FIXED

Description

Calling the function `updateConfig` in the `StakingHandler` contract can disrupt its work. This is possible for the following reasons:

- There is no validation of `_weight` values. `_weight` can be equal to `0` and break the calculation of `share` in `streams` for staking holders. This will result in incorrect calculation of the repayment of staked tokens and rewards when exiting the stacking, which will block the work of the contract.
- Updating the `voteToken` parameter will cause the contract to try to burn new `voteToken` tokens that are not on the balance when `unlock` is called.
- Updating the parameters `rewardsCalculator`, `voteShareCoef`, `maxLockPeriod`, `maxLockPositions` will also lead to incorrect calculations and contract blocking.

Recommendation

We recommend discarding the `updateConfig` function and consider mechanisms for stacking migration to a new contract with a suspension of the contract work during migration, e.g. `emergencyExit`.

Update

Fathom's response

Implemented `emergencyUnlockAndWithdraw` applicable when contract is paused.

Oxorio's response

The fix was implemented, but with a small flaw - there is no check on how much `lock` the user has. If the user has `0` `lock`, the loop will be skipped and the execution will continue until the `payRewards` function is called with `0` tokens, which will cause a revert.

We recommend adding a check that the user's `lock` number is greater than `0`.

Fathom's response

The fix was implemented

2.2.1 In `MultiSigWallet` there's no parameter defining minimum amount of signatures

SEVERITY	**MAJOR**
STATUS	**FIXED**

Description

The parameter `_numConfirmationsRequired` is checked in the constructor and in the function `changeRequirement`, that is not equal to `0`, however, when multi-signature is set, it allows the value `1`, and the contract may be used by one of the `owners`.

Recommendation

We recommend adding minimum quantity constant for necessary signatures, e.g. `MIN_CONFIRMATIONS` and check if the set value is greater than or equal to `MIN_CONFIRMATIONS`.

Update

Fathom's response

Implemented Auditors Recommendation.

```
modifier validRequirement(uint ownerCount, uint _required) {
        require(
            ownerCount > 0 && ownerCount <= MAX_OWNER_COUNT && _required <= ownerCount &&
ownerCount > 1 ? _required > 1 : _required > 0,
            "MultiSig: Invalid requirement"
        );
        _;
}
```

Oxorio's response

The fix was implemented by adding a check to the `validRequirement` modifier applicable to functions, but no changes were made in the constructor.

We recommend adding a check to the constructor as well.

Fathom's response

The fix was implemented

2.2.2 Transaction does not have a lifetime parameter in `MultiSigWallet`

SEVERITY	MAJOR
STATUS	FIXED

Description

In the structure `Transaction` there's no lifetime parameter `expired`, which is responsible for the period of time during which the transaction must be executed. Since transactions may be executed at random time and are not removed over time, frozen, previously not approved transactions can be executed after a certain time and cause an undesirable effect.

Recommendation

We recommend adding an individual parameter, which is responsible for the maximum time until the transaction can be executed, e.g. `expired` and check it before running transactions.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The recommendation is not implemented correctly.
We meant the `lifetime` parameter, which is passed as a function parameter.

`lifetime` must be greater than the minimum value and already be in the body of the function to get the value.

```
transactions[_txIndex].expireTimestamp = block.timestamp + lifetime
```

Fathom's response

Implemented Auditors Recommendation.

Lifetime parameter added, with the criteria that if the lifetime is zero then it can be executed anytime in the future, plus a `MAX_LIFETIME` Parameter added.

Oxorio's response

The fix was implemented in `MultiSigWallet#L181`.

2.2.3 Governance can delete `TimelockAdmin` and the contract will lose its control in `TimelockController`

SEVERITY	**MAJOR**
STATUS	**FIXED**

Description

In the `TimelockController` contract, Governance can take away the `TIMELOCK_ADMIN_ROLE` rights from the address `admin`. In the case of an attack on `Governance` and `Council` this would make it impossible to revoke the role from the captured contracts.

Recommendation

We recommend to consider a permissions policy or add the `DEFAULT_ADMIN_ROLE` for `admin` to be able to revoke the role in case of an attack.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The recommendation was not fully implemented.
Admin role was added but not functions like `grantRole` and `revokeRole` for specific roles from the list of possible ones on behalf of `DEFAULT_ADMIN_ROLE`.
Only `TIMELOCK_ADMIN_ROLE` can change or delete `TIMELOCK_ADMIN_ROLE`, if the role was

deleted from admin, then even having the `DEFAULT_ADMIN_ROLE` role, will not work with the built-in external functions of the `AccessControl` contract.

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in `TimelockController#L228-L234`.

2.2.4 There is no validation for `maxTargets` when executing in `Governor`

SEVERITY	**MAJOR**
STATUS	**FIXED**

Description

In the `Governor` contract in the `propose` function there is no validation of the maximum number of `targets`. This can cause `proposal` to have so many calls to external contracts that the execution transaction will face a "gas bomb" effect. This means a large amount of gas consumption or restricted gas limit block.

Recommendation

We recommend including the `maxTargets` parameter for `_targets`, the maximum number of `_targets` in the `proposal`.

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in commit with identifier 2c40cfad6436ed2a9d9563213e4db222aae31f5e.

2.2.5 There is no possibility to update `multisig` in `Governor`

SEVERITY	MAJOR
STATUS	FIXED

Description

In the `Governor` contract there is no possibility to perform a migration to a new `multisig`. For example to a new version of the contract.

Recommendation

We recommend adding the `updateMultisig` function, but so that only the old `multisig` could call it.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in commit with identifier 2c40cfad6436ed2a9d9563213e4db222aae31f5e.

2.2.6 There is no emergency shutdown mode in `Governor`

SEVERITY	MAJOR
STATUS	FIXED

Description

There is no possibility in the `Governor` contract to put it into an emergency shutdown status. If one of the `TimelockController`, `MultiSigWallet` contracts is compromised,

Governance will not be able to perform an emergency shut-down of proposals execution and stop contracts.

Recommendation

We recommend adding the `emergencyExit` function to the contract, which can be called by Governance by majority vote without confirmation with `multisig`. The function can be called once, its call stops the work of the contract. After calling this function, recovery is only possible by migrating to a new contract.

Update

Fathom's response

Implemented Auditors Recommendation. and added `emergencyStop`

Oxorio's response

An `emergencyStop` method has been added, but the problem still remains.

- The method just calls the `pause()` function
- The method is called on behalf of `Multisig`, which can be compromised.

The main idea of this function is to put the contract into an emergency exit state, which can only be restored by completely replacing the contract and the states. This is an extreme case, an emergency stop. There should be no possibility to `unpause` after `emergencyStop` call.
We propose refactoring this code according to the recommendation.

Fathom's response

Implemented Auditors Recommendation and added emergencyStop.

Oxorio's response

The fix was implemented in `MainTokenGovernor#L86-L99`.

2.2.7 It is possible to set a null address in `GovernorTimelockControl` when updating `timelock`

SEVERITY	**MAJOR**
STATUS	**FIXED**

Description

In the `GovernorTimelockControl` contract it is possible to set a null address when calling the function `_updateTimelock`. This can make the execution of `proposals` not possible since it is done through `timelock`. It will be also not possible to recover or change `timelock`, since it needs the corresponding proposal to be executed, which is also not possible with a zero `timelock`.

Recommendation

We recommend adding a check that the address `newTimelock != address(0)`

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

A redundant validation in the constructor in `GovernorTimelockControl`:

```
if (address(timelockAddress) == address(0)) {
    revert ZeroAddress();
}
```

We recommend removing it because the same validation can be found in `_updateTimelock`.

Fathom's response

The fix was implemented

2.2.8 There is no validation for null values for `newQuorumNumerator` in `GovernorVotesQuorumFraction`

SEVERITY	**MAJOR**
STATUS	**FIXED**

Description

In the `GovernorVotesQuorumFraction` contract in the `_updateQuorumNumerator` function it is possible to set `_quorumNumerator` to `0` value, which would lead to a complete voting stop.

Recommendation

We recommend adding a constant with the minimum allowable value of `_quorumNumerator` and perform a corresponding check in the `_updateQuorumNumerator` function.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in commit with identifier 2c40cfad6436ed2a9d9563213e4db222aae31f5e.

2.2.9 When `MINTER_ROLE` is added to `VMainToken`, the `isWhiteListed` list does not update

SEVERITY	**MAJOR**
STATUS	**FIXED**

Description

In the VMainToken contract, for mint tokens, calling account, in addition to having `MINTER_ROLE` rights, must also be in the `isWhiteListed` list, since the mint function calls `_mint,` which contains `_beforeTokenTransfer` call.
When `_beforeTokenTransfer` is called, it checks that the `msg.sender` address is in the `isWhiteListed` list.
In the case of `mint`, it is the address with the `MINTER_ROLE` rights.
The administrator can grant/revoke `MINTER_ROLE` from an address by calling `grantRole`/`revokeRole`, but the `isWhitelisted` list remains unchanged - the old address stays in the list while the new one is never added.
This creates a risk that if `MINTER_ROLE` is compromised by an attacker, the admin will not be

able to correctly revoke his rights, and the attacker can make a `transfer` of tokens to unauthorized addresses.

Recommendation

We recommend adding separate functions to grant and revoke the `MINTER_ROLE`, which will also add and remove addresses from the `isWhitelisted` list.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

- When revoking minter rights, `_grantRole` is used instead of `_revokeRole`. It should be changed to:

```
function revokeMinterRole(address _minter) public override onlyRole(getRoleAdmin(MINTER_ROLE))
{
    _revokeRole(MINTER_ROLE, _minter);
```

- To add and remove from whitelist the following functions are used:

```
   function addToWhitelist(address _toAdd) public override onlyRole(WHITELISTER_ROLE) {
        isWhiteListed[_toAdd] = true;
        emit MemberAddedToWhitelist(_toAdd);
    }

    function removeFromWhitelist(address _toRemove) public override onlyRole(WHITELISTER_ROLE)
{
        isWhiteListed[_toRemove] = false;
        emit MemberRemovedFromWhitelist(_toRemove);
    }
    function grantMinterRole(address _minter) public override
onlyRole(getRoleAdmin(MINTER_ROLE)){
        _grantRole(MINTER_ROLE, _minter);

        addToWhitelist(_minter);
    }

    function revokeMinterRole(address _minter) public override
onlyRole(getRoleAdmin(MINTER_ROLE)){
        _grantRole(MINTER_ROLE, _minter);
```

```
        removeFromWhitelist(_minter);
    }
```

But it should be noted that `addToWhitelist` and `removeFromWhitelist` can be called from `WHITELISTER_ROLE`. In this case, `MINTER_ROLE` must also have `WHITELISTER_ROLE`.

We recommend refactoring this code and adding internal functions `_addToWhitelist` and `_removeFromWhitelist` without access control to `grantMinterRole` and `revokeMinterRole`.

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in `VMainToken#L46-L54`.

2.2.10 There is no possibility to transfer standard `ERC20` tokens from the Governance balance in `MainTokenGovernor`

SEVERITY	**MAJOR**
STATUS	**FIXED**

Description

In the `MainTokenGovernor` contract there is no possibility to transfer tokens of the `ERC20` standard from the balance of Governance, because execution of the transaction is actually passed to the `TimelockController`.

Recommendation

We recommend fixing the possibility of withdrawal of tokens of the `ERC20` standard from the balance of Governance. This can be done in the following way:

- It is a must to implement the `addSupportingTokens` function due to the fact that various tokens of the `ERC20` standard can be transferred to the Governance balance. Governance must work only with trusted tokens like USDT, USDC, etc. This function will make it possible to create a list of trusted tokens. Adding a token should only be done through Governance.

- Add a check to the `execute` function to confirm that `_target` is the contract address from the trusted tokens. And only in this case pass it to the `TimelockController` address.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The `relay` function is implemented incorrectly.

```
   function relay(address target, uint256 value, bytes calldata data) external payable virtual
onlyGovernance {
        require(isSupportedToken[target],"relay: token not supported");
        (bool success, bytes memory returndata) = target.call{value: value}(data);
        Address.verifyCallResult(success, returndata, "Governor: relay reverted without
message");
    }
```

Now it is possible to send value to a supported token contract. In this case all value sent to the token contract will be lost.
We recommend making two different functions for relaying `ERC20` tokens and native coins, e.g. `relayERC20` and `relayETH`.

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in `MainTokenGovernor#L148-L156`.

2.2.11 There is no option to migrate to another contract in the `VaultPackage` contract

SEVERITY	**MAJOR**
STATUS	**FIXED**

Description

The `VaultPackage` contract lacks the ability to suspend a contract in an emergency and migrate assets to a new compatible `VaultPackage` contract.

Recommendation

We recommend adding the `emergencyExit` function in the contract which permanently blocks contract function calls for `REWARD_OPERATOR_ROLE`, and adding the `migrate` function, which allows to move tokens and token balances to a new version of `VaultPackage`.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The migration flow is not complete.

- After migration, `Vault` can still be used.
  We recommend forbidding to use functions after migration.
- At the `VaultPackage#L89` `migrate` function is using balance of the `Vault` tokens instead of `deposited` mapping. In this case, during the migration, the tokens that got into the contract by accident will become deposited tokens of the new `Vault` and will be used as rewards.

We recommend using `deposited` variable instead `balanceOf` `VaultPackage` balance.

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in `VaultPackage.sol#L122`.

2.2.12 There is a DoS possibility when calling `updateVault` in the `StakingHandlers` contract

SEVERITY	**MAJOR**
STATUS	**FIXED**

Description

In the `StakingHandlers` contract, calling the function `updateVault` can cause all contract functions that work with balances and `VaultPackage` functions to be blocked.

Recommendation

We recommend improving this function in the following way:

- The `VaultPackage` update must be available if the current `VaultPackage` is put into `emergencyExit` status (see recommendation to this issue).
- Updating `VaultPackage` must only take place after calling the `migrate` function in the old `VaultPackage`.
- Updating `VaultPackage` must only take place if the migration of balances to the new `VaultPackage` was successful.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The recommendation has not been fully implemented.

```
function updateVault(address _vault) public override onlyRole(DEFAULT_ADMIN_ROLE) {
    // enforce pausing this contract before updating the address.
    // This mitigates the risk of future invalid reward claims
    require(paused != 0, "require pause");
    require(_vault != address(0), "zero addr");
    require(IVault(vault).migrated(), "nt migrated");
    vault = _vault;
}
```

Despite checking that the `vault` is migrated, there is no validation that `_vault` is a compatible `VaultPackage`, which is the contract where the migration took place.
We recommend adding new statement that `_vault` is `VaultPackage` for migration.

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in `8cdb8eac2916c7b45731a2c672a7601e5b022cb6` commit.

2.2.13 There is no emergency suspension of the rewards payment in the `VaultPackage` contract

SEVERITY	MAJOR
STATUS	FIXED

Description

In the `VaultPackage` contract there is no possibility to suspend the function `payRewards`. This causes the attacker to continue taking tokens from the contract if the address with `REWARDS_OPERATOR_ROLE`, such as `StakingHandlers` contract, is compromised.

Recommendation

We recommend adding the `pausable` modifier to the `payRewards` function of the `VaultPackage` contract.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in commit with identifier 2c40cfad6436ed2a9d9563213e4db222aae31f5e.

2.2.14 Unsafe use of the `transfer` and `transferFrom` functions in `StakingHandlers` and `VaultPackage`

SEVERITY	MAJOR
STATUS	FIXED

Description

In the `StakingHandlers` and `VaultPackage` contracts there are unsafe `transfer` and `transferFrom` functions of the `ERC20` standard. The use of these functions is not recommended as not all tokens clearly comply with the `ERC20` standard, more details here.

Recommendation

We recommend using the `SafeERC20` extension from the OpenZepplin library and replace the `transfer` and `transferFrom` calls with `safeTransfer` and `safeTransferFrom`.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in commit with identifier 2c40cfad6436ed2a9d9563213e4db222aae31f5e.

2.2.15 Tokens that get into the `VaultPackage` balance can be used to withdraw rewards in the contract `StakingHandler`

SEVERITY	**MAJOR**
STATUS	**FIXED**

Description

In the `VaultPackage` contract tokens that get into the balance of the contract can be used for rewards payment from streams in StakingHandlers. This results in tokens, that get on the balance by mistake and/or intentionally, not being able to be withdrawn from the contract.

Recommendation

We recommend:

- adding a separate `deposit` function in the `VaultPackage` contract and make reward payments through the `deposited` parameter.
- adding a separate `withdraw` function that would allow the `DEFAULT_ADMIN_ROLE` address to take excess tokens away (both `supportedTokens` and tokens that are not on the list).
- replacing token transfers to `VaultPackage` in the `StakingHandlers` contract with calling the `deposit` function of the `VaultPackage` contract. It should have a prior `safeApprove` call to token in the `VaultPackage` contract.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

Recommendation has not been fully implemented. In the current version there is still no possibility to withdraw tokens that got into the contract by accident.
We recommend adding a separate withdraw function, that would allow the `DEFAULT_ADMIN_ROLE`
address to take excess tokens away (both `supportedTokens` and tokens that are not on the list).

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The recommendation is implemented.

2.2.16 Calling `initializeStaking` in the `StakingHandlers` contract does not allocate rewards for `MAIN_STREAM` in `VaultPackage`

SEVERITY	**MAJOR**
STATUS	**FIXED**

Description

In the `StakingHandlers` contract the `initializeStaking` function does not allocate tokens for rewards `MAIN_STREAM`, as it happens when `createStream` is called. This may result in the block of the `withdrawStream` function call from the `MAIN_STREAM` of tokens and rewards for some users, if the amount in `VaultPackage` is less than the amount stated in `scheduleRewards`.

Recommendation

We recommend moving the initialization of `MAIN_STREAM` from `initializeStaking`, that can be called when creating `StakingProxy`, to the `initializeMainStream` function, which can only be called by `STREAM_MANAGER_ROLE`. Before calling this function the work of the contract must be suspended.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The implementation of the recommendation has led to new problems. `initializeMainStream` can be reinitialized in `StakingHandlers`. `initializeMainStream` function is missing custom initialize modifier in order to prevent it from the reinitialization. Any manager with `STREAM_MANAGER_ROLE` can create a stream without proposing it.
We recommend adding custom `stakingInitializer` modifier in order to prevent future reinitializations of the main stream.

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in `StakingHandler#L66`.

2.2.17 Updating `rpsDuringLastClaimForLock` for inactive `stream` in the `StakingInternals` contract

SEVERITY	MAJOR
STATUS	FIXED

Description

In the `StakingInternals` contract when the `_stake` function is called the calculation of `rpsDuringLastClaimForLock` is done even for inactive `streams`. This can lead to both excessive gas consumption and denial of service if the number of `streams`, active and inactive, is too large.

Recommendation

We recommend adding a check that the `stream`, for which the check takes place, has `ACTIVE` status.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in commit with identifier 2c40cfad6436ed2a9d9563213e4db222aae31f5e.

2.2.18 There is a possibility for a manager to remove all streams in order to steal all pending rewards in `StakingHandlers`

SEVERITY	MAJOR
STATUS	FIXED

Description

In the contract `StakingHandlers` in the `removeStream` function a manager can remove `stream` with pending rewards for users. This will result in users losing their pending rewards.

Recommendation

We recommend adding logic to check that there are no pending rewards for users in the `stream` before it can be deleted.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in commit with identifier 2c40cfad6436ed2a9d9563213e4db222aae31f5e.

2.2.19 `MINTER_ROLE` and `WHITELISTER_ROLE` have the same value in the `VMainToken`

SEVERITY	MAJOR
STATUS	FIXED

Description

In the contract `VMainToken` the `MINTER_ROLE` and `WHITELISTER_ROLE` constants have the same value:

```
bytes32 public constant MINTER_ROLE = keccak256("MINTER_ROLE");
bytes32 public constant WHITELISTER_ROLE = keccak256("MINTER_ROLE");
```

When the role is set, the `WHITELISTER_ROLE` variable will in fact be set to the `MINTER_ROLE`. This will result in the user getting both roles and an address with `WHITELISTER_ROLE` being able to call the `mint` and `burn` functions.

Recommendation

We recommend updating the setting of `WHITELISTER_ROLE` constant:

```
bytes32 public constant WHITELISTER_ROLE = keccak256("WHITELISTER_ROLE");
```

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in commit with identifier 2c40cfad6436ed2a9d9563213e4db222aae31f5e.

2.2.20 Transaction should be marked as `executed` if the call fails

SEVERITY	**MAJOR**
STATUS	**FIXED**

Description

In the contracts:

- `MultiSigWallet.sol#L137-L145)`
- `TimelockController.sol#L111`
- `Governor.sol#L76`

If the call fails, all the state changes of the contract will be reverted. It means that this call would not be marked as `executed` and can be repeated in the future, since it has enough confirmations.

Recommendation

We recommend marking transaction as `executed` in all cases, removing lines with statement of revert failed transactions, and adding `data` value to event.

Update

Fathom's response

Implemented Auditors Recommendation.

For `TimelockController`, it makes sense to revert on fail of execute as it will make sure that the bad proposals are not marked executed if it fails.

Oxorio's response

Recommendation was not implemented.
In the contracts

- `MultiSigWallet.sol#L232-L236`
- `TimelockController.sol#L226`
- `Governor.sol#L430`

the failed call will lead to all the state changes of the contract to be reverted. It means that this call would not be marked as `executed` and can be repeated in the future, since it has enough confirmations. This can lead to unexpected behavior, the state of the blockchain could be changed and already executed failed transaction could be re-executed and be successful.
As for `TimelockController`, the revert on fail of `_execute` does not mark the proposal as bad proposal, e.g. if the call has logic connected with timestamps it may be reverted on the one block and be successful on the next block.

We recommend marking transaction as `executed` in all cases, removing lines with statement of reverting the failed transactions, and adding `data` value to the event. If the status of the call is `false`, transaction should not be reverted.

Fathom's response

Since MultiSig is used only by trusted wallets, this is not necessary functionality.

Oxorio's response

The fix was implemented in contracts:

- `TimelockController#L242`.
- `Governor.sol#L458`.
  But wasn't implemented in `MultiSigWallet#L226-L230`.

2.2.21 Admin role can be revoked forever by mistake in `VMainToken`

SEVERITY	MAJOR
STATUS	FIXED

Description

In the contract `VMainToken` in the `initToken` function, the value of `admin` can be the same as `msg.sender` and thus it becomes possible that an `admin` accidently revokes admin role from himself.

Recommendation

We recommend adding a check that `admin` is not equal to `msg.sender`.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in commit with identifier 2c40cfad6436ed2a9d9563213e4db222aae31f5e.

2.2.22 It is possible for attacker to create active locks to force users to reach the lock limit in `StakingHandlers`

SEVERITY	MAJOR
STATUS	FIXED

Description

In the `StakingHandler` contract the attacker can create active locks for token holders with `createLockWithoutEarlyWithdraw` function by using max value for `lockPeriod` in multiple transactions. In this case user's locks limit can be reached and they will not be able to enter the staking until the end of the lock period.

Recommendation

We recommend:

1. Revising the logic of the `createLock` and `createLockWithoutEarlyWithdraw` functions and making a separate limit for creating a lock from a third-party address.
2. Or creating a lock from the `msg.sender` address.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in commit with identifier 2c40cfad6436ed2a9d9563213e4db222aae31f5e.

2.2.23 `prohibitedEarlyWithdraw` is not set to `false` for `lockid` after unlocking in `StakingHandlers`

SEVERITY	**MAJOR**
STATUS	**FIXED**

Description

In the function `createLockWithoutEarlyWithdraw` in the `StakingHandlers` contract parameter `prohibitedEarlyWithdraw` for given `lockid` is set to `true`, but it does not update to `false` after unlocking later in the `unlock` and `unlockPartially` functions. Since the value in the `locks` array is deleted after the unlock, all new values will be assigned the value of `prohibitedEarlyWithdraw`, regardless of whether the `createLockWithoutEarlyWithdraw` or `createLock` function is called.

Recommendation

We recommend setting `prohibitedEarlyWithdraw[account][lockId]` to `false` before deleting value from `locks` array in the `unlock` and `unlockPartially` functions:

```
prohibitedEarlyWithdraw[msg.sender][lockId] = false;
```

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

```
prohibitedEarlyWithdraw[msg.sender][lockId] = false;
```

There is no setting of `prohibitedEarlyWithdraw` to `false` for the `unlockPartially` method.

At the same time, it can be found in the `earlyUnlock` method, but it is not needed there since this method only works when the value is already set to `false`.
We recommend adding `prohibitedEarlyWithdraw` to `unlockPartially` and removing it from `earlyUnlock` functions.

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in `StakingHandler#225`.

2.2.24 Calling `unlock`, `earlyUnlock` and `unlockPartially` before `claimRewards` will result in loss of rewards in `StakingHandlers`

SEVERITY	**MAJOR**
STATUS	**NO ISSUE**

Description

In the contract `StakingHandlers` the following functions can cause a loss of rewards if they are called before `claimRewards`:

- `unlock`
- `earlyUnlock`
- `unlockPartially`

It is possible because:

- `unlock` and `earlyUnlock` functions contain an internal call to the `_unlock`, where `lock` with given `lockId` is removed
- in `unlockPartially` the `rpsDuringLastClaimForLock` for given `lockId` is updated

As a result, rewards for given `lockId` will be lost.

Recommendation

We recommend adding internal function `_claimRewards` and claim rewards with the calls to `unlock`, `earlyUnlock`, and `unlockPartially` functions.

Update

Fathom's response

Frontend is designed in a way that tells the user to claim all the rewards before unlocking it. So we accept the risk of rewards loss if the user ignores this notification.
You can try it on dapp.fathom.fi.

2.2.25 Share weight drop formula is incorrect in `StakingInternals`

SEVERITY	MAJOR
STATUS	FIXED

Description

In the `StakingInternals` contract share weight drop formula is incorrect:

```
uint256 shares = amountOfTokenShares + (voteShareCoef * nVoteToken) / 1000;
uint256 slopeStart = streams[MAIN_STREAM].schedule.time[0] + ONE_MONTH;
uint256 slopeEnd = slopeStart + ONE_YEAR;
if (timestamp <= slopeStart) return shares * weight.maxWeightShares;
if (timestamp >= slopeEnd) return shares * weight.minWeightShares;
return
    shares *
    weight.maxWeightShares +
    (shares * (weight.maxWeightShares - weight.minWeightShares) * (slopeEnd - timestamp)) /
    (slopeEnd - slopeStart);
```

It appears that the weight of the shares should gradually fall over time from `weight.maxWeightShares` to `weight.minWeightShares`.

However, the current formula implements a weight drop from (2* `weight.maxWeightShares` - `weight.minWeightShares`) to `weight.maxWeightShares`.

Recommendation

We recommend changing `weight.maxWeightShares` to `weight.minWeightShares` in weight drop formula:

```
return
    shares *
    weight.minWeightShares +
    (shares * (weight.maxWeightShares - weight.minWeightShares) * (slopeEnd - timestamp)) /
    (slopeEnd - slopeStart);
```

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in commit with identifier 2c40cfad6436ed2a9d9563213e4db222aae31f5e.

2.2.26 Penalty can be bigger than stake in the `StakingInternals`

SEVERITY	**MAJOR**
STATUS	**FIXED**

Description

In the contract `StakingInternals` there is a penalty calculation in the `_earlyUnlock` function:

```
uint256 penalty = (weighingCoef * amount) / 100000;
user storage userAccount = users[account];
userAccount.pendings[MAIN_STREAM] -= penalty;
```

The maximum value of the `weightingCoef` that it can take is `weight.penaltyWeightMultiplier * weight.maxWeightPenalty`. In this case, the weight parameters are not checked in any way during initialization. If they are set in a way that the product of `weight.penaltyWeightMultiplier * weight.maxWeightPenalty` is greater than `100000`, then the penalty will be greater than the amount, which in turn will lead to excessive pendings or overflow.

Recommendation

We recommend adding the following check to `initializeStaking` and `updateConfig`:

```
require(weight.penaltyWeightMultiplier * weight.maxWeightPenalty <= 100000, "Wrong penalty weight");
```

It is also worth moving the value of `100000` into a separate constant variable to improve the readability of the code.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

```
require(weight.penaltyWeightMultiplier * weight.maxWeightPenalty <= 100000, "wrong weight");
```

The value of `weight` is checked before setting `weight = _weight`, so the result of multiplying `weight.penaltyWeightMultiplier * weight.maxWeightPenalty` will always be `0`.

We recommend replacing the validation to:

```
require(_weight.penaltyWeightMultiplier * _weight.maxWeightPenalty <= 100000, "wrong weight");
```

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in `StakingInternals#L41`.

2.3.1 Modifier `onlyOwnerOrGov` creates a complex confirmation structure in case of `Governance` calls in the `MultiSigWallet`

SEVERITY	**WARNING**
STATUS	**NO ISSUE**

Description

The modifier `onlyOwnerOrGov` uses the following construction:

```
require(isOwner[msg.sender] || governor == msg.sender, "MultiSig: MultiSigWallet,
onlyOwnerOrGov(): Neither owner nor governor");
```

that allows calling the following functions in the contract on behalf of Governance:

- `submitTransaction`
- `confirmTransaction`
- `revokeConfirmation`

However, `Governance` may commit contract calls only with permission from `MultiSigWallet`.

The result is that, if `Governance` wants to call a transaction on a `MultiSigWallet` contract:

- `Governance` creates `proposal` for a call to `MultiSigWallet`.
- `MultiSigWallet` after confirmation by owners must call `confirmProposal` on `Governance`.
- Then `Governance` may call one of `MultiSigWallet` functions.
- In this case, however, `MultiSigWallet` transaction execution still requires signature of `owners`.

Schematically, is looks like the following:

- To make a call for `MultiSigWallet` it takes steps: `Governance` -> `createProposal` -> `confirmProposal`.

- To execute `confirmProposal` it takes steps: `MultiSigWallet` -> `submitTransaction` -> `confirmTransaction` -> `executeTransaction`.
- To make a call for `MultiSigWallet` it requires the next steps from `Governance`: `Governance` -> `execute` -> `MultiSigWallet`.

And so each function in the sequence:

- `submitTransaction`
- `confirmTransaction`
- `revokeConfirmation`

Recommendation

We recommend removing `Governance` from this modifier and give the permission to `MultiSigWallet` administration to authorized representatives only, or review the logic of `Governance` and approving of `proposals` from `MultiSigWallet`.

Update

Fathom's response

Thats the way its designed

2.3.2 No parameter check when adding transaction in `MultiSigWallet`

SEVERITY	**WARNING**
STATUS	**FIXED**

Description

In the function `submitTransaction` there's no validation of address `_to` to be the contract. Based on the logic of the contract, there may be the following cases:

- `_to` is a `EOA` address, `_value != 0`, `_data = ""`.
- `_to` is a contract.

Recommendation

We recommend adding parameter checking when adding a transaction according to possible cases of using `MultiSigWallet`.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The recommendation is not implemented correctly.

```
if (_to.isContract()) {
    require(_data.length > 0, "no calldata for contract call");
} else {
    require(_data.length == 0 && _value > 0, "calldata for EOA call or 0 value");
}
```

This implementation prohibits transferring `ETH` to the contract's balance. Since in the current condition it is assumed that if `_to` is a contract, then `_data` must not be empty.

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in `MultiSigWallet#L130-L134`, but with logic that is misleading, since if revert occurs because `_to` is EOA and `_data.length > 0`, an `InsufficientValue` error is thrown even though the original reason is calldata for EOA.

We recommend creating a separate custom error to be used when the `_data.length > 0` condition is true, so that the real reason for the revert can be understood.

Fathom's response

The fix was implemented. We changed error revert name.

2.3.3 Missing validation, that the bytecode of address `_to` did not change while running a transaction in `MultiSigWallet`

SEVERITY	**WARNING**
STATUS	**FIXED**

Description

In the functions `confirmTransaction` and `executeTransaction` there's no validation that the bytecode of address `_to` did not change as an EOA or smart contract.
In this case, the following situations are possible:

- when the transaction was added with the parameter `_to` as an EOA address, i.e. with an empty bytecode, and when the transaction is executed, frontrunning may occur and the attacker may deploy to `_to` address a smart contract with malicious code, using metamorphic contracts and `create2` opcodes.
- when the transaction was added with the parameter `_to` as a smart contract, and at the moment of transaction execution, frontrunning may occur, and the attacker may change the bytecode at the `_to` address for a smart contract with malicious code using metamorphic contracts and `create2` opcodes.

Recommendation

We recommend adding:

- checking that `_to` is an EOA address and when `confirmTransaction` and `executeTransaction` if the contract isn't deployed into the adress, using `isContract` from OpenZeppelin.
- checking that the contract's bytecode has not been changed, recording the bytecode hash into a separate mapping, e.g.:

```
bytes32 codeHash;
assembly {
    codeHash = extcodehash(_to);
}

isWhitelistedBytesCode[_to] = codeHash;

...
bytes32 codeHash;
assembly { codeHash := extcodehash(account) }

return (codeHash != isWhitelistedBytesCode[_to]);
```

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

Corrections were made, but one fact that was not taken into consideration is that the `_to` hash is overwritten in the `submitTransaction` function and the following scenario may occur:

- The owner or `Governor` is proposed to call the contract at `0xA` address, proposal number 1.
- The desired number of votes is gained.
- The contract `0xA` changes the bytecode using the metamorphic technique.
- One of the owners or the `Governor` offers another call to contract `0xA`, which overwrites the stored code hash for `0xA` and after that proposal number 1 is able be executed even though the code of `0xA` has changed.

Alternatively, consider another situation where one of the owners is malicious and offers to call a smart-contract that is under his ownership, and in the same scenario as described above, changes the logic before the call itself.

We recommend refactoring the logic of the code and to take the described scenario into account.

Fathom's response

We implemented mapping to use txn Index to have more granular approach.

2.3.4 There's no ETH balance validation when adding a non-zero transaction `_value` in `MultiSigWallet`

SEVERITY	**WARNING**
STATUS	**FIXED**

Description

In the function `submitTransaction` there's no verifying that `MultiSigWallet` account has the necessary amount on the balance for the transaction. In case of approval by `owners`, the transaction will be approved but not executed.

Recommendation

We recommend adding balance check while adding a transaction with a non-zero value `_value`.

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in commit with identifier 2c40cfad6436ed2a9d9563213e4db222aae31f5e.

2.3.5 There is no time limit for executing proposal in `Governor`

SEVERITY	WARNING
STATUS	FIXED

Description

The `Governor` contract has no parameters for the time limit on `proposal` execution. This can result in no longer relevant proposal being executed after a period of time.

Recommendation

We recommend adding the `lifetime` parameter, the runtime of `proposal`, and check it during the execution.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

We have not found a implemented corrections for this issue.
We recommend adding a lifetime parameter, the runtime of proposal, and check it during the execution.

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in `Governor.sol#L222`.

2.3.6 There is no check for gas consumption in `Governor`

SEVERITY	WARNING
STATUS	NO ISSUE

Description

In the Governor contract, the `propose` function lacks a parameter and a check for gas limit for calls to `targets`. This could make it possible for a call to a vulnerable external contract to be able to loop the call and perform a DDoS attack with high gas consumption.

Recommendation

Consider implementing the `gasLimit` parameter - the maximum gas amount for a call, for each of the `targets`.

Update

Fathom's response

We will have voting for proposal and multisig execution confirmation. Thats hard to DDoS there, so we won't implement gas check.

2.3.7 `confirmProposal` is possible for both active and inactive proposals in `Governor`

SEVERITY	WARNING
STATUS	FIXED

Description

In the `Governor` contract the function `confirmProposal` can be called for both active and inactive proposals.

Recommendation

We recommend adding a check that the proposal is either successful or already scheduled in the `confirmProposal` function:

```
ProposalState status = state(proposalId);
require(status == ProposalState.Succeeded || status == ProposalState.Queued, "Governor:
proposal not successful");
```

Update

Fathom's response

Implemented Auditors Recommendation.

2.3.8 There is no check for the `msg.value` value available for execution in `Governor` and `TimelockController`

SEVERITY	**WARNING**
STATUS	**FIXED**

Description

In the `Governor` and `TimelockController` contracts the `execute` functions do not check the `msg.value` balance value needed to execute `_targets`, which would result in gas consumption even if the amount of `ETH` is not enough.

Recommendation

We recommend adding:

- a check that the `msg.value` passed to the `execute` function is greater than the total value needed for the execution of the `targets` calls in the proposal.
- a return of the remaining `ETH` balance to the sender of the transaction after the execution of `proposal`.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The value of the transferred ETH is checked in the `TimelockController` contract when the `execute` method is executed, but is not checked for `executeBatch`, which is actually used

in the contracts.
We propose refactoring this code according to the recommendation.

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in `TimelockController#L165`.

2.3.9 There is no check for zero value for `_token`, `_multiSig` and `_timelock` in `Governor`, `GovernorTimelockControl`, `MainTokenGovernor`

SEVERITY	**WARNING**
STATUS	**FIXED**

Description

In the constructors of `Governor`, `GovernorTimelockControl` and `MainTokenGovernor` contracts it is possible to set zero values for `tokenAddress`, `_multiSig`, `timelock` contracts.

This may cause that `_token`, `_multiSig` and `_timelock` can be set to a zero address by mistake and break the contract. Thus, it will not be possible to update these parameters because an update is only possible from `Governance`, and `Governance` will cannot update parameters if `_timelock` is zero.

Recommendation

We recommend adding a validation that the `_token`, `_multiSig`, `_timelock` addresses in the constructor are not zero.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in

2.3.10 There is no check for zero in `GovernorSettings._setProposalThreshold`

SEVERITY	WARNING
STATUS	FIXED

Description

In the `_setProposalThreshold` function it is possible to set `_proposalThreshold` to `0`. This can lead to a proposer be able to create a proposal with no voting tokens on the balance, or with a minimum number of them (e.g. `1 wei`). This creates a DDoS attack threat.

Recommendation

We recommend adding a check that `newProposalThreshold` is not zero.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in

2.3.11 There is no limit on the number of proposals for one proposer in `Governor`

SEVERITY	WARNING
STATUS	FIXED

Description

In the `Governor` contract in the `propose` function there is no limit on the number of proposals for one proposer. Thus, a proposer can perform a DDoS attack and create an unlimited number of requests, even in one single block.

Recommendation

We recommend adding a limit to the number of proposals with `active` and `pending` status.

Update

Fathom's response

```
nextAcceptableProposalTimestamp[msg.sender] = block.timestamp + proposalTimeDelay;
```

Oxorio's response

The implemented fix does not fully resolve the problem.
The Proposer can still create an unlimited number of proposals.
We recommend adding a limit for pending proposals for one user.

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was not implemented, blacklist was added for DDoS protection in `Governor#L202`.

But the important function `setBlocklistStatusForProposer`, which changes the status of the user to blocked, has no event, making it impossible for the UI to keep track of the situation of blocked users.

We recommend adding an event that will be emitted when the user is blocked.

Fathom's response

We added the required event

2.3.12 A missing check that tokens are on the balance when calling the `payRewards` function in the `VaultPackage` contract

SEVERITY	**WARNING**
STATUS	**FIXED**

Description

In the `VaultPackage` contract when calling the function `payRewards` there is no processing of errors such as:

- There is no check that tokens are on the balance.
- There is no check that the value of `amount != 0`.

Recommendation

We recommend adding a check that tokens are on the balance and that `amount != 0`, and return error using `custom errors` (`revert CustomError`) or with `require`.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in

2.3.13 There is no limit on the maximum number of active `streams` in the `StakingHandlers` contract

SEVERITY	**WARNING**
STATUS	**NO ISSUE**

Description

In the `StakingHandlers` contract there is no limit on the maximum number of active `streams`. This creates a situation of an uncontrolled gas consumption when dealing with contract functions and can lead to DoS.

Recommendation

We recommend adding a parameter that would allow to limit the maximum number of active `streams`.

Update

Fathom's response

This will be handled by Stream Manager.

Oxorio's response

Although it was evident that `STREAM_MANAGER_ROLE` was not a completely secure address, there have been a number of recent cases when a particular role could be compromised.
We strongly recommend to consider adding appropriate features and validations as described earlier.

2.3.14 Incorrect processing of contract modifiers `Initializable` in the `StakingHanders` contract

SEVERITY	**WARNING**
STATUS	**FIXED**

Description

The contract `StakingHandlers` uses the `upgradeable proxy` template, at the same time the work with the modifiers of the `Initializable` contract, which is inherited from the `AdminPausable`, is not performed correctly.

Recommendation

We recommend adjusting the contract according to OpenZeppelin's recommendations:

- The contract constructor must contain a call to the `_disableInitializers` function to disable contract initialization at the implementation level and prevent an attacker from using the contract's implementation
- The initializer (in the case of the `StakingHandlers` contract it is `initializeStaking`) must contain the `initializer` modifier
- The initialiser of the parent contract must be with the `onlyInitializing` modifier (in the case of the `StakingHandlers` contract, it is a call to the `pausableInit` of the `AdminPausable` contract)

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in commit with identifier 2c40cfad6436ed2a9d9563213e4db222aae31f5e.

2.3.15 It is possible for any user to call `createStream` in the `StakingHandlers` contract

SEVERITY	**WARNING**
STATUS	**FIXED**

Description

In the `StakingHandlers` contract any user can call the function `createStream` and run `stream`. This bears a risk that attackers could mislead a potential user into giving `approve` to the `StakingHandlers` contract and force them to call `createStream`. `createStream` will charge the user the necessary amount of money for the rewards.

Recommendation

We recommend adding a condition that `createStream` can only be called from the `streamOwner` address.

Update

Fathom's response

Implemented Auditors Recommendation.

2.3.16 Possible overflow with calculations

SEVERITY	**WARNING**
STATUS	**FIXED**

Description

In the next lines there is a possible overflow:

- `RewardsLibrary.sol#L70`
- `RewardsLibrary.sol#L71`
- `RewardsLibrary.sol#L78`
- `RewardsLibrary.sol#L8`
- `RewardsCalculator.sol#L70`
- `RewardsCalculator.sol#L77`
- `RewardsCalculator.sol#L83`
- `RewardsInternals.sol#L15`
- `RewardsInternals.sol#L24-L25`
- `StakingInternals.sol#L47`
- `StakingInternals.sol#L45`
- `StakingInternals.sol#L227-L230`

Recommendation

We recommend to use `muldiv` to multiply elements safely.
We also recommend to update `voteLockCoef` initialization and add checks that it is not zero (to prevent division by zero) and that it is not too big in order to avoid overflow in `BoringMath`.

Update

Fathom's response

Done where feasible for contract size

Oxorio's response

We recommend fixing these issues completely, if there is already a problem with the size of the contract, then the code needs to be refactored.

Fathom's response

Implemented Auditors Recommendation for all except for these:

- RewardsLibrary.sol#L70
- RewardsLibrary.sol#L71
- RewardsLibrary.sol#L78
- RewardsLibrary.sol#L8
- RewardsCalculator.sol#L70 [DONE]
- RewardsCalculator.sol#L77 [DONE]
- RewardsCalculator.sol#L83 [DONE]
- RewardsInternals.sol#L15
- RewardsInternals.sol#L24-L25
- StakingInternals.sol#L47
- StakingInternals.sol#L45
- StakingInternals.sol#L227-L230
- StakingInternals.sol#L45

Our total Supply is 1 billion. Even if we have 100 billion total supply, the above line will not overflow as,

```
nVoteToken = (amount * lockPeriod * POINT_MULTIPLIER) / voteLockCoef / POINT_MULTIPLIER

nVoteToken = 100 * 1e9(amount) * 1e18 * 1e9(lock period) * 1e18(Point Multiplier) / 500
(VoteLockCoef)/ 1e18 (Point Multiplier)
           ~= approx 1e57,
But since nVoteToken in appr.1e77, it will not overflow
```

- StakingInternals.sol#L47

I converted uint128 to uint256 for voteTokenBalance here,

https://github.com/Into-the-Fathom/fathom-dao-smart-contracts/blob/reaudit-fixes-final/contracts/dao/staking/packages/StakingInternals.sol#L59

- StakingInternals.sol#L227-L230

This will not overflow as maxWeightShares, minWeightShares are always less than 1e9 at max.

- RewardsInternals.sol#L15

This will not overflow as:

```
streams[streamId].schedule.reward[i] = (streams[streamId].schedule.reward[i] * rewardTokenAmount) / streams[streamId].maxDepositAmount
```

If One Billion was the amount of reward Token It will Be:
`1e9 * 1e18 * 1e9 * 1e18 / (1e9 * 1e18)`
Which is,

1e54 is the upper limit which is again less than 2^256

- RewardsInternals.sol#L24-L25

For this:

```
uint256 reward = ((streams[streamId].rps - userAccount.rpsDuringLastClaimForLock[lockId]
[streamId]) * lock.positionStreamShares) /
            RPS_MULTIPLIER;
```

RPS_MULTIPLIER changed to 1e36. So max upper limit is 1e36 * 1e9 = approx. 1e45
https://github.com/Into-the-Fathom/fathom-dao-smart-contracts/blob/reaudit-fixes-final/contracts/dao/staking/StakingStorage.sol#L13

Which is again less than 1e77

For others fix is applied.

We accept this issue for the above described lines as calculations shows there is no overflow.

Oxorio's response

The fix was implemented in contracts:

- RewardsLibrary#L75
- RewardsLibrary#L82
- RewardsLibrary#L87
- RewardsCalculator#L74
- RewardsCalculator#L81
- RewardsCalculator#L87
- StakingInternals#L66

2.3.17 Multiple `streams` can be active at the same time with the same parameters in `StakingHandler.sol`

SEVERITY	WARNING
STATUS	NO ISSUE

Description

In the contract StakingHandler it is possible to add and activate `streams` with the same parameters. This can lead to duplicate `streams` with the same parameters executed by mistake.

Recommendation

We recommend adding checks that `stream` is added before submitting a new one.

Update

Fathom's response

This will be handled by Stream Manager.

Oxorio's response

Although it was evident that `STREAM_MANAGER_ROLE` was not a completely secure address, there have been a number of recent cases when a particular role could be compromised.
We strongly recommend to consider adding appropriate features and validations as described earlier.

2.3.18 There is no limit for the amount of schedules on streams in `StakingHandlers`

SEVERITY	WARNING
STATUS	NO ISSUE

Description

There is no limit for the amount of schedules on streams in the contract `StakingHandlers`. This can cause the block gas limit to be exceeded.

Recommendation

We recommend limiting values of `scheduleTimes` or `scheduleRewards`.

Update

Fathom's response

This will be handled by Stream Manager.

Oxorio's response

Although it was evident that `STREAM_MANAGER_ROLE` was not a completely secure address, there have been a number of recent cases when a particular role could be compromised. We strongly recommend to consider adding appropriate features and validations as described earlier.

2.3.19 It is possible to remove tokens that are used by another contract in `VaultPackage`

SEVERITY	**WARNING**
STATUS	**FIXED**

Description

Calling the `removeSupportedToken` function in the `VaultPackage` contract removes tokens which are used in the `StakingHandler` contract to pay rewards and staked tokens.

Recommendation

We recommend adding logic to check that tokens are not used in any other contract before removing them.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in commit with identifier 2c40cfad6436ed2a9d9563213e4db222aae31f5e.

2.4.1 There's no logging of reverted transactions in `MultiSigWallet`

SEVERITY	INFO
STATUS	FIXED

Description

In the function `executeConfirmation` there's no logging of failed transactions.

```
(bool success, ) = transaction.to.call{ value: transaction.value }(transaction.data);
require(success, "tx failed");
```

Recommendation

We recommend replace this construction for the next one:

```
error TransactionRevered(bytes data);
...
(bool success, bytes data) = transaction.to.call{ value: transaction.value }(transaction.data);

if (success) {
    emit ExecuteTransaction(msg.sender, _txIndex);
} else {
    revert TransactionRevered(data);
}
```

This will allow monitoring of suspicious activity that involves using of `MultiSigWallet`.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

```
if (success) {
    emit ExecuteTransaction(msg.sender, _txIndex);
} else {
    revert TransactionRevered(data);
}

emit ExecuteTransaction(msg.sender, _txIndex);
```

Two identical `ExecuteTransaction` events will be emitted on successful execution of the transaction. We recommend removing one from the `MultiSigWallet#L218`.

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in `MultiSigWallet#L229`.

2.4.2 Non-optimal packing of the `Transaction` structure in `MultiSigWallet`

SEVERITY	**INFO**
STATUS	**FIXED**

Description

The structure `Transaction` uses a non-optimized storage layout.

Recommendation

We recommend optimizing storage layout the following way:

```
struct Transaction {
    address to;
    bool executed;
    bytes data;
    uint value;
```

```
    uint numConfirmations;
}
```

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in commit with identifier 2c40cfad6436ed2a9d9563213e4db222aae31f5e.

2.4.3 Incorrect status check in `execute` function in `Governor`

SEVERITY	**INFO**
STATUS	**FIXED**

Description

In the `execute` function there is an incorrect check of `Proposal` status:

```
require(status == ProposalState.Succeeded || status == ProposalState.Queued, "Governor:
proposal not successful");
```

In the MainTokenGovernor.sol contract, that inherits from `Governor`, the execution is passed to the `TimelockController` contract. For a transaction to be executed through `TimelockController` it must only have the `ProposalState.Queued` status. Otherwise the gas will be wasted and the `execute` call will be reverted.

Recommendation

We recommend changing the status check for `Proposal`:

```
require(status == ProposalState.Queued, "Governor: proposal not successful");
```

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in commit with identifier 2c40cfad6436ed2a9d9563213e4db222aae31f5e.

2.4.4 `_minDelay` can be set to zero in `TimelockController`

SEVERITY	**INFO**
STATUS	**FIXED**

Description

In the `TimelockController` contract the `_minDelay` parameter can be set to `0` during initialization and in the `updateDelay` function. This will result in batch being able to be executed in the same block it was queued for execution.

Recommendation

We recommend adding a check that `_minDelay != 0`.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in commit with identifier 2c40cfad6436ed2a9d9563213e4db222aae31f5e.

2.4.5 There is a redundant `initialized` check in `VMainToken`

SEVERITY	INFO
STATUS	FIXED

Description

```
require(!initialized, "already init");
initialized = true;
```

The `initToken` function contains redundant code with checking and setting the value of the `initialized` parameter, since this check already exists in the `initializer` modifier in the `initToken` function.

Recommendation

We recommend deleting these lines.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in commit with identifier 2c40cfad6436ed2a9d9563213e4db222aae31f5e.

2.4.6 There is redundant code in the `VMainToken` contract

SEVERITY	INFO
STATUS	FIXED

Description

The `_mint` and `_burn` functions in the `VMainToken` contract are redundant and essentially do not overload the parent functions.

Recommendation

We recommend deleting these functions.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fixes have been made <>, but there is still one function with redundant code: `_afterTokenTransfer`, because it too does not change the behavior of the inherited contract and can be deleted.

2.4.7 The `Governor` and `TimeLockController` do not support the `ERC721` and `ERC1155` tokens

SEVERITY	**INFO**
STATUS	**NO ISSUE**

Description

The `Governor` and `TimelockController` contracts lack the following methods:

```
/**

 * @dev See {IERC721Receiver-onERC721Received}.
 */
function onERC721Received(
    address,
    address,
    uint256,
    bytes memory
) public virtual override returns (bytes4) {
```

```
    return this.onERC721Received.selector;
}

/**

 * @dev See {IERC1155Receiver-onERC1155Received}.
 */
function onERC1155Received(
    address,
    address,
    uint256,
    uint256,
    bytes memory
) public virtual override returns (bytes4) {
    return this.onERC1155Received.selector;
}

/**

 * @dev See {IERC1155Receiver-onERC1155BatchReceived}.
 */
function onERC1155BatchReceived(
    address,
    address,
    uint256[] memory,
    uint256[] memory,
    bytes memory
) public virtual override returns (bytes4) {
    return this.onERC1155BatchReceived.selector;
}
```

Thus `Governor` and `TimeLockController` do not support tokens with `ERC721` and `ERC1155` standards.

Recommendation

We recommend implementing these functions if the `Governor` and `TimeLockController` contracts require support for the `ERC721` and `ERC1155` tokens. And also create a list of trusted tokens that can work with (see above - `ERC20` standard tokens transfer possibility).

Update

Fathom's response

There is no provision for `ERC721` and `ERC1155` tokens to be deposited into the contract.

2.4.8 The `addSupportedToken` and `removeSupportedToken` calls have an redundant `pausable` modifier in the `VaultPackage` contract

SEVERITY	**INFO**
STATUS	**FIXED**

Description

In the `VaultPackage` contract the calls `addSupportedToken` and `removeSupportedToken` have a redundant modifier `pausable` since the calls are only possible from the `DEFAULT_ADMIN_ROLE` address and the modifier `pausable` contains the following condition

```
require((paused & flag) == 0 || hasRole(DEFAULT_ADMIN_ROLE, msg.sender), "paused contract");
```

where the `paused` condition will be ignored.

Recommendation

We recommend reconsidering the `addSupportedToken` and `removeSupportedToken` function modifiers or removing the `pausable` modifier.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in commit with identifier 2c40cfad6436ed2a9d9563213e4db222aae31f5e.

2.4.9 There are no checks that `admin`, `proposers` and `executors` are not zero addresses in `TimelockController`

SEVERITY	**INFO**
STATUS	**FIXED**

Description

In the contract `TimelockController` constructor there are no checks that `admin`, `proposers` and `executors` are not zero addresses.

Recommendation

We recommend adding checks that `admin`, `proposers` and `executors` are not zero addresses.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in commit with identifier 2c40cfad6436ed2a9d9563213e4db222aae31f5e.

2.4.10 Unused import of `StakingStructs` in `StakingStorage`

SEVERITY	**INFO**
STATUS	**FIXED**

Description

Import of `StakingStructs` in the `StakingStorage` contract is never used.

Recommendation

We recommend removing it to keep the codebase clean.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in commit with identifier 2c40cfad6436ed2a9d9563213e4db222aae31f5e.

2.4.11 Unused constant `ONE_MONTH` in `StakingGettersHelper`

SEVERITY	**INFO**
STATUS	**FIXED**

Description

The `ONE_MONTH` constant in the `StakingGettersHelper` contract is never used.

Recommendation

We recommend removing it to keep the codebase clean.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in commit with identifier 2c40cfad6436ed2a9d9563213e4db222aae31f5e.

2.4.12 Non-optimal storage layout for `Stream` struct in `StakingStructs`

SEVERITY	**INFO**
STATUS	**FIXED**

Description

`Stream` struct in the `StakingStructs` contract has non-optimal storage layout.

Recommendation

We recommend moving `StreamStatus` definition after the `rewardToken` line in the struct `Stream` in order to store values in one slot.

```
struct Stream {
    address owner; // stream owned by the ERC-20 reward token owner
    address manager; // stream manager handled by Main stream manager role
    address rewardToken;
    StreamStatus status;
    uint256 rewardDepositAmount; // the reward amount that has been deposited by a third party
    uint256 rewardClaimedAmount; /// how much rewards have been claimed by stakers
    uint256 maxDepositAmount; // maximum amount of deposit
    uint256 minDepositAmount; // minimum amount of deposit
    uint256 tau; // pending time prior reward release
    uint256 rps; // Reward per share for a stream j>0
    Schedule schedule;
}
```

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in commit with identifier 2c40cfad6436ed2a9d9563213e4db222aae31f5e.

2.4.13 Unnecessary `'` in a `RewardsLibrary` comment

SEVERITY	INFO
STATUS	NO ISSUE

Description

There is an explicit `'` in the comment in `RewardsLibrary.sol#L82` line.

Recommendation

We recommend removing `'` from the comment.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

Library was removed at commit with identifier 2c40cfad6436ed2a9d9563213e4db222aae31f5e

2.4.14 There is a typo in a comment in `StakingInternals`

SEVERITY	INFO
STATUS	FIXED

Description

There is a typo in the word "have" in the following line `StakingInternals.sol#L95`.

```
// user does not hae enough voteToken, it is still able to burn and unlock
```

Recommendation

We recommend changing it to:

```
// user does not have enough voteToken, it is still able to burn and unlock
```

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in `StakingInternals#L119`.

2.4.15 Redundant check for `maxDepositAmount > 0` in `RewardsCalculator`

SEVERITY	**INFO**
STATUS	**FIXED**

Description

There is a redundant check for `maxDepositAmount > 0` in the next lines:

- RewardsCalculator.sol
- RewardsLibrary.sol

Since `minDepositAmount` is already greater than `0` and `maxDepositAmount` must be bigger than `minDepositAmount` there is no need to check that `maxDepositAmount > 0`.

Recommendation

We recommend removing requirement of `maxDepositAmount > 0` for gas savings and improving code readability.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The check remains in the `RewardsLibrary#L21`:

```
require(maxDepositAmount > 0, "No Max Deposit");
```

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in RewardsLibrary.sol#L20-L24.

2.4.16 It is not possible to withdraw tokens that were sent by mistake

SEVERITY	INFO
STATUS	NO ISSUE

Description

It is not possible to withdraw tokens that were sent by mistake it the following contracts:

- RewardsCalculator.sol
- StakingPackage.sol
- VMainToken.sol
- MainToken.sol

Recommendation

We recommend adding `sweep` function to withdraw tokens that were sent by mistake.

Update

Fathom's response

There is no provision of tokens being sent in those contract.

2.4.17 Unused import of `ReentracyGuard` in `StakingHandlers`

SEVERITY	**INFO**
STATUS	**FIXED**

Description

There is import of `ReentracyGuard` in the `StakingHandlers` contract but `nonReentrant` from this class is never used in `StakingHandlers`.

Recommendation

We recommend removing the unused import.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in commit with identifier 2c40cfad6436ed2a9d9563213e4db222aae31f5e.

2.4.18 Custom `initializer` modifier is used instead of one from OpenZeppelin

SEVERITY	**INFO**
STATUS	**FIXED**

Description

It is better to use Openzeppelin `initializer` instead of custom modifiers in the next functions:

- `StakingHandler.sol#L33`
- `VaultPackage.sol#L18`

- VMainToken.sol#L24

Recommendation

We recommend using `initializer` and `initializable` modifiers from Openzeppelin instead of implementing custom modifiers.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The recommendation has not been fully implemented.

```
require(!vaultInitialized, "Vault: Already Initialized");
vaultInitialized = true;
```

The `vaultInitialized` variable becomes meaningless after adding the `initializer` modifier to the `initVault` function in `VaultPackage` contract.
In `VMainToken` contract `initToken` function uses `initializer`, additional `bool` variable `initialized` was not removed.

We recommend removing custom `initializer` variables and validations.

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in contracts:

- VaultPackage#L14.
- VMainToken#L12.

2.4.19 Stream manager, treasury manager and admin represent the same account in `StakingHandlers`

SEVERITY	INFO
STATUS	NO ISSUE

Description

In the `initializeStaking` function in the `StakingHandlers` contract multiple roles are assigned to the same `admin` address.

Recommendation

We recommend to transfer treasury role after the deployment and the staking setting. Admin and manager of the initial `stream` should be two different roles.

Update

Fathom's response

This is initial setup to make it easier. We will share roles after some time.

2.4.20 Revert message strings are too long

SEVERITY	INFO
STATUS	NO ISSUE

Description

- `VMainToken.sol#L65-L68`
- `MultiSigWallet#L30`
- `MultiSigWallet.sol#L55`
- `MultiSigWallet.sol#L77`

After the revert message string is split into 32-byte sized chunks and stored in `memory` using `mstore`, the `memory` offsets are given to `revert(offset, length)`. For chunks shorter than 32 bytes, and for low `--optimize-runs` values (usually even the default value of `200`), instead of using `push32(val)` (where `val` is the 32 byte hexadecimal representation of the string with zero padding on the least significant bits) the Solidity compiler replaces it by `shl(value, short-value)`, where `short-value` does not have any zero padding. This saves the total amount of bytes in the deploy code and therefore saves deploy time cost, at the expense of extra 6 gas consumption during runtime.
This means that shorter revert strings saves deploy time costs of the contract. Note that this is not relevant for high values of `--optimize-runs` since `push32` value will not be replaced by a `shl(value, short-value)` equivalent by the Solidity compiler.

Going back, each 32 byte chunk of the string requires an extra `mstore`. That is, additional cost for `mstore`, `memory` expansion costs, as well as stack operations. Note that this runtime cost is only relevant when the revert condition is met.

Overall, shorter revert strings can save deploy time as well as runtime costs.

Recommendation

We recommend making revert strings shorter.
Note that if your contracts already allow Solidity `0.8.4` and above, then consider using custom errors. They provide more gas efficiency and also allow developers to describe the errors in detail using NatSpec. The main disadvantage of this approach is that some tooling may not have proper support for it yet.

Update

Fathom's response

Not Done, right now. Lots of changes for revert strings might be required right now.

2.4.21 Unnecessary reads from storage

SEVERITY	INFO
STATUS	NO ISSUE

Description

In the next lines using `MLOAD` and `MSTORE` to cache the variable in `memory` saves more gas than `SLOAD`, since they use only 3 gas, instead of the initial 100:

- `MultiSigWallet.sol#L138`
- `StakingHandler.sol#L191`
- `StakingHandler.sol#L200`
- `StakingHandler.sol#L210`
- `StakingHandler.sol#L237`
- `StakingHandler.sol#L244`

Recommendation

We recommend caching this storage variable in `memory` to reduce unnecessary reads from storage and save more gas.

Update

Fathom's response

Not Done, increases contract size.

2.4.22 Misleading check `(scheduleTimeLength > 0)` in the `RewardsCalculator`

SEVERITY	**INFO**
STATUS	**FIXED**

Description

In the function `_getStartEndScheduleIndex` in the contract `RewardsCalculator` there is the following condition:

```
require(scheduleTimeLength > 0, "bad schedules");
```

This condition allows `scheduleTimeLength` value to be set to 1. This can lead to underflow and incorrect operation of cycles further down the code.

Recommendation

We recommend changing it to

```
require(scheduleTimeLength >= 2, "bad schedules");
```

or completely remove this check, since this condition is already checked in `validateStreamParameters()` when the stream is created.

Update

Fathom's response

Implemented Auditors Recommendation.

Oxorio's response

The fix was implemented in commit with identifier 2c40cfad6436ed2a9d9563213e4db222aae31f5e.

3 CONCLUSION

The following table contains the total number of issues that were found during audit:

Level	Amount
CRITICAL	8
MAJOR	26
WARNING	19
INFO	22
Total	**75**

All issues were fixed as part of the current reaudit. In addition to the remarks in the reaudit, the code base was even changed.
We recommend performing a full-fledged audit of the actual version of the code.

THANK YOU FOR CHOOSING

OXORIO