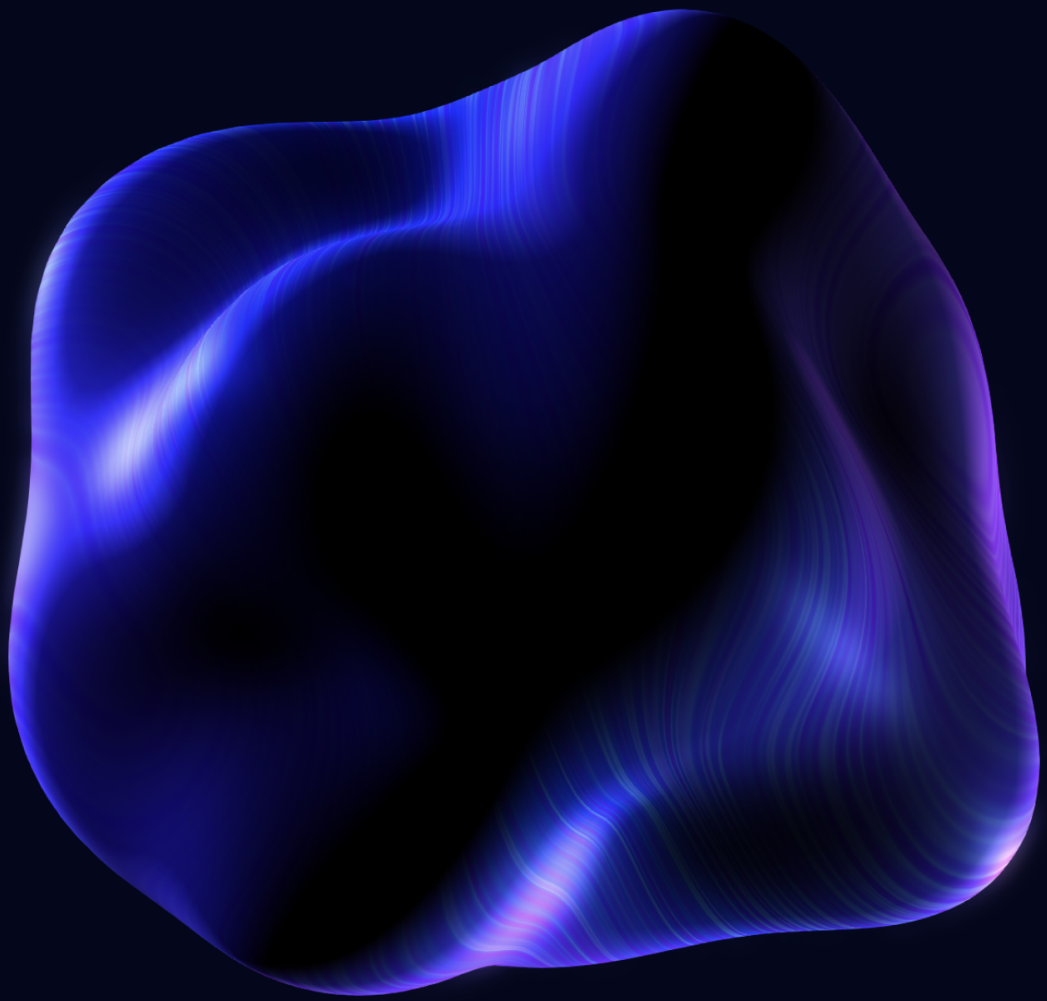
MUNDUS
SECURITY

Security
Smart Contract Audit



MUNDUS
SECURITY

MAHADao Governance security audit

This document may contain confidential information about IT systems and the intellectual property of the Customer as well as information about potential vulnerabilities and methods of their exploitation.
The report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities are fixed — upon a decision of the Customer.

Reference information

Name	MahaDAO Governance Contracts
Language	Solidity
Chain	Ethereum mainnet
Website	https://mahadao.com/
Documentation	https://docs.mahadao.com/
Reference repositories	https://github.com/MahaDAO/governance-contracts https://github.com/MahaDAO/token

MUNDUS
SECURITY

Scope of work

Contract	Address
BaseV2Bribes	0x8f362e16a74c2eb564bfbf24dc73bd5ce37d9063
BaseV2Voter *Proxy*	0x227a445ff220cc9c3584fe77b7dfef6af0b63e8e 0xeb99748e91afca94a6289db3b02e7ef4a8f0a22d
EmissionController	0xbd86a195c90cec4606dbc378ea0aa338f674a704
FeesSplitter	0x9032f1bd0cc645fde1b41941990da85f265a7623
GaugeLP *Proxy*	0xd2125a722d28c7685aed658a3ddc7b08275b8aeb 0x9ee8110c0aacb7f9147252d7a2d95a5ff52f8496
GaugeUniswapV3 *Proxy*	0xa7af7eaa2bf2fbea3fdb90db1a820508ed3f037c 0x98e1701f6558dd63481b57926c9f22c64d918c35
MAHATimelockController-14	0x43c958affe41d44f0a02ae177b591e93c86adbea
MAHATimelockController-30	0xb45021f5313b93927699aae6cbe989bccf6b5900
MahaToken	0x745407c86df8db893011912d3ab28e68b62e49b0
MAHAXGovernor	0xe7d23c2b3e9148c46cec796f018842ab72d5867f
MAHAXLocker	0xbdd8f4daf71c2cb16cce7e54bb81ef3cfcf5aacb
MAHAXStaker	0x608917f8392634428ec71c6766f3ec3f5cc8f421
MAHAXVetoGovernor	0x9a7e7b4c2abe3255dec67e3bf2e6b24b46223111
Registry	0x2684861ba9dada685a11c4e9e5aed8630f08afe0
RenderingContract	0x9d348281e16218cd8ede9cd8a1bca74e89b410e8

MUNDUS
SECURITY

Table of contents

1. Findings summary
2. Source code audit
3. Disclaimers

MUNDUS
SECURITY

Findings summary

Finding Severity breakdown

All vulnerabilities discovered during the source code audit are classified based on their potential severity and have the following classification:

Severity	Description
High	Bugs that can trigger a contract failure or theft of assets. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss of funds.
Low	Bugs that do not pose significant danger to the project or its users but are recommended to be fixed nonetheless.
Informational	All other non-essential recommendations.
Gas	Gas optimization recommendations.

Findings statistics

Severity	Number
High	0
Medium	2
Low	5
Informational	13
Gas	9
Total	**29**

MUNDUS
SECURITY

Findings

ID	Severity	Description
01	**Medium**	distributeETH should be non-reentrant in FeesSplitter.sol
02	**Medium**	Arbitrary IERC20 in distributeERC20 public of FeesSplitter.sol
03	Low	Unnecessary events emitted by getReward in BaseV2Bribes.sol
04	Low	Sanity check required for setters in Registry.sol
05	Low	Sanity check required for period initialization in Epoch.sol
06	Low	Sanity check required for amountToSplit in FeesSplitter.sol
07	Low	Sanity check required for _accounts.length in FeesSplitter.sol
08	Informational	Unused imports in GaugeLP.sol
09	Informational	Unused imports in BaseV2Voter.sol
10	Informational	Unused import {INFTLocker} in BaseV2Bribes.sol
11	Informational	onlyOwner modifier for checkPercentages method is excessive in FeesSplitter.sol
12	Informational	Misleading comment in MAHAXVetoGovernor.sol
13	Informational	Misleading comment in MAHAXGovernor.sol
14	Informational	Misleading comment in GaugeLP.sol
15	Informational	Misleading comment in EmissionController.sol
16	Informational	Misleading comment in BaseV2Bribes.sol
17	Informational	Logic duplication in getReward and getRewardForOwner in BaseV2Bribes.sol
18	Informational	Dead code: SafeERC20 in FeesSplitter.sol
19	Informational	Dead code: modifier checkStartTime in Epoch.sol
20	Informational	Add events to MAHAXLocker.sol

ID	Severity	Description
21	Gas	`_users.length` should be cached in for-loops in `MAHAXLocker.sol`
22	Gas	`tokens.length` should be cached in for-loops in `BaseV2Bribes.sol`
23	Gas	`tokenIds.length` should be cached in for-loops in `GaugeUniswapV3.sol`
24	Gas	`targets.length` should be cached in for-loops in `MAHATimelockController.sol`
25	Gas	`registry` should be immutable in `MAHAXLocker.sol`
26	Gas	`registry` should be immutable in `EmissionController.sol`
27	Gas	`_percentAllocations.length` should be cached in for-loops in `FeesSplitter.sol`
28	Gas	`_gauges.length` should be cached in for-loops in `BaseV2Voter.sol`
29	Gas	`balanceOf[account]` should be cached in for-loops in `GaugeUniswapV3.sol`

MUNDUS
SECURITY

Source code audit

ID-01. **Medium**: distributeETH should be non-reentrant in FeesSplitter.sol

Description

The distributeETH method of FeesSplitter.sol performs low-level calls to addresses in a for-loop to distribute ETH. This pattern is prone to reentrancy, e.g. a CREATE2 address with predefined malicious logic could be provided to accounts state variable which later drains out the **FeesSplitter** contract.

Recommendation

Add nonReentrant modifier to the distributeETH method of FeesSplitter.sol.

ID-02. **Medium**: Arbitrary IERC20 in distributeERC20 public of FeesSplitter.sol

Description

The distributeERC20 method of FeesSplitter.sol has no access control and takes arbitrary ERC20 as its argument. This is a dangerous pattern that should be addressed.

Recommendation

Modify the distributeERC20 method's logic in any or all of the following ways

- Add nonReentrant and/or onlyOwner modifier to the distributeERC20 method.
- Use Openzeppelin's SafeERC20 library for token transfers.
- Introduce ERC20 whitelist to the **FeesSplitter** contract's storage.

ID-03. Low: Unnecessary events emitted by `getReward` in `BaseV2Bribes.sol`

Description

The `getReward` and `getRewardForOwner` methods of `BaseV2Bribes.sol` transfers rewards per each `tokens[i]` if `_reward > 0`. However, the `ClaimRewards` event is emitted irrespective of reward for a particular token being `> 0`.

Recommendation

Construct a `_getReward()` `internal` function (see `Issue`) and modify its logic in the following way

```
if (_reward > 0) {
    _safeTransfer(tokens[i], _owner, _reward);
    emit ClaimRewards(_owner, tokens[i], _reward);
}
```

ID-04. Low: Sanity check required for setters in `Registry.sol`

Description

The setters for state variables of type `address` in `Registry.sol`, e.g. the `setMAHA` method, should exercise `isContract` sanity check.

ID-05. Low: Sanity check required for `period` initialization in `Epoch.sol`

Description

Each epoch timestamp in `Epoch.sol` is calculated as the product of epoch number, `_getNextEpoch()`, and the epoch period, `period`. Subsequently, other methods execute their corresponding logic via comparing this product with `block.timestamp`. Thus, the `period` cannot be less than the maximum time delta between consequent blocks. Adding a sanity check for `period` being larger than some reasonable time delta is required.

Recommendation

Add `require(_period >= 86400)` to the constructor and `setPeriod` method of `Epoch.sol`.

ID-06. Low: Sanity check required for `amountToSplit` in `FeesSplitter.sol`

Description

Sanity check for `amountToSplit > PERCENTAGE_SCALE/min(percentAllocations)` is required for the `distributeETH` and `distributeERC20` methods of `FeesSplitter.sol`.

Recommendation

Adding `require(amountToSplit > PERCENT_SCALE)` to the `distributeETH` and `distributeERC20` methods of `FeesSplitter.sol` should suffice.

ID-07. Low: Sanity check required for `_accounts.length` in `FeesSplitter.sol`

Description

Sanity check is required for lengths of `accounts` and `percentAllocations` state variables in the constructor and `updateSplit` method of `FeesSplitter.sol`.

Recommendation.

Add `require(_accounts.length == _percentAllocations.length)` to constructor and `updateSplit` method of `FeesSplitter.sol`.

ID-08. Informational: Unused imports in `GaugeLP.sol`

Description

`IBribe` and `INFTLocker` are unused imports in `GaugeLP.sol` and should be removed.

ID-09. Informational: Unused imports in BaseV2Voter.sol

Description

IBribeFactory and IUniswapV2Pair are unused imports in BaseV2Voter.sol and should be removed.

ID-10. Informational: Unused import {INFTLocker} in BaseV2Bribes.sol

Description

INFTLocker is an unused import in BaseV2Bribes.sol and should be removed.

ID-11. Informational: onlyOwner modifier for checkPercentages method is excessive in FeesSplitter.sol

Description

The onlyOwner modifier for function checkPercentages view is excessive as it does not deal with contract's storage on its own. The onlyOwner modifier should be removed and the checkPercentages method should be declared pure.

ID-12. Informational: Misleading comment in MAHAXVetoGovernor.sol

Description

MAHAXVetoGovernor.sol contains misleading comment on the quorum method (see @audit).

```
/**                              @audit
 * @dev Returns the quorum for a block number,
 * in terms of number of votes: `supply * numerator / denominator`.
 */
function quorum(uint256 blockNumber)
    public
    view
    override
    returns (uint256)
{
    return _quorum; // @audit
}
```

ID-13. Informational: Misleading comment in MAHAXGovernor.sol

Description

MAHAXGovernor.sol contains misleading comment on the quorum(uint256) method (see @audit).

```
/**                              @audit
  * @dev Returns the quorum for a block number,
  * in terms of number of votes: `supply * numerator / denominator`.
  */
function quorum(uint256) public view override returns (uint256) {
    return _quorum; // @audit
}
```

ID-14. Informational: Misleading comment in GaugeLP.sol

Description

GaugeLP.sol contains the same comment on supplyCheckpoints and rewardPerTokenCheckpoints state variables.

```
/// @notice A record of balance checkpoints for each token, by index
mapping(uint256 => SupplyCheckpoint) public supplyCheckpoints;

/// @notice A record of balance checkpoints for each token, by index
mapping(address => mapping(uint256 => RewardPerTokenCheckpoint))
    public rewardPerTokenCheckpoints;
```

ID-15. Informational: Misleading comment in EmissionController.sol

Description

The allocateEmission method of EmissionController.sol contains misleading comment about token transfer approval, which doe not take place in this method. This is likely related to the code base upgrade from BaseV1Voter.sol to BaseV2Voter.sol.

```
// approve token and notify the gauge voter
IERC20(registry.maha()).transfer(registry.gaugeVoter(), balanceToSend);
IGaugeVoter(registry.gaugeVoter()).notifyRewardAmount(balanceToSend);
```

ID-16. Informational: Misleading comment in BaseV2Bribes.sol

Description

GaugeLP.sol contains the same comment on supplyCheckpoints and rewardPerTokenCheckpoints state variables.

```
/// @notice A record of balance checkpoints for each token, by index
mapping(uint256 => SupplyCheckpoint) public supplyCheckpoints;

/// @notice A record of balance checkpoints for each token, by index
mapping(address => mapping(uint256 => RewardPerTokenCheckpoint))
    public rewardPerTokenCheckpoints;
```

ID-17. Informational: Logic duplication in getReward and getRewardForOwner in BaseV2Bribes.sol

Description

The getReward and getRewardForOwner methods of BaseV2Bribes.sol are unnecessary duplicates of one another. Consider defining a _getReward internal function and getReward and getRewardForOwner wrappers for it.

ID-18. Informational: Dead code: SafeERC20 in FeesSplitter.sol

Description

OpenZeppelin's library SafeERC20 is imported but never used in FeesSplitter.sol.

ID-19. Informational: Dead code: modifier checkStartTime in Epoch.sol

Description

checkStartTime is an unused modifier in Epoch.sol and should be removed.

ID-20. Informational: Add events to MAHAXLocker.sol

Description

The setRoyaltyInfo and setMinLockAmount methods of MAHAXLocker.sol should emit corresponding events.

ID-21. Gas: _users.length should be cached in for-loops in MAHAXLocker.sol

Description

_users.length should be cached in the uploadUsers method of MAHAXLocker.sol.

ID-22. Gas: tokens.length should be cached in for-loops in BaseV2Bribes.sol

Description

tokens.length should be cached for gas savings in getReward and getRewardForOwner methods of BaseV2Bribes.sol.

ID-23. Gas: tokenIds.length should be cached in for-loops in GaugeUniswapV3.sol

Description

tokenIds.length should be cached for gas savings in the isIdsWithinRange and claimFeesMultiple methods of GaugeUniswapV3.sol.

ID-24. Gas: `targets.length` should be cached in for-loops in `MAHATimelockController.sol`

Description

`targets.length` should be cached for gas savings in the `scheduleBatch` and `executeBatch` methods of `MAHATimelockController.sol`.

ID-25. Gas: `registry` should be immutable in `MAHAXLocker.sol`

Description

`registry` state variable of `MAHAXLocker.sol` should be immutable for gas savings.

ID-26. Gas: `registry` should be immutable in `EmissionController.sol`

Description

`registry` state variable of `EmissionController.sol` should be immutable for gas savings.

ID-27. Gas: `_percentAllocations.length` should be cached in for-loops in `FeesSplitter.sol`

Description

`_percentAllocations.length` should be cached for gas savings in the `checkPercentages` method of `FeesSplitter.sol`.

ID-28. Gas: `_gauges.length` should be cached in for-loops in `BaseV2Voter.sol`

Description

`_gauges.length` should be cached for gas savings in `distribute(address[] memory)` and `updateFor(address[] memory` methods of `BaseV2Voter.sol`.

ID-29. Gas: `balanceOf[account]` should be cached in for-loops in `GaugeUniswapV3.sol`

Description

`balanceOf[account]` should be cached for gas savings in the `getReward(address account, address[] memory)` method of `GaugeUniswapV3.sol`.

Disclaimers

Mundus disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Technical disclaimers

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.