Ouroboros Minter Security Audit

Report Version 1.1

October 13, 2024

Conducted by **Hunter Security**:

Bounty Hunter, Senior Security Auditor
byter0x1, Senior Security Auditor

Table of Contents

1 About Hunter Security

Hunter Security is an industry-leading smart contract security auditing firm. Having conducted over 100 security audits protecting over $1B of TVL, our team always strives to deliver top-quality security services to the best DeFi protocols. For security audit inquiries, you can reach out on Telegram or Twitter at *@georgehntr*.

2 Disclaimer

Audits are a time-, resource-, and expertise-bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can reveal the presence of vulnerabilities **but cannot guarantee their absence**.

3 Risk classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

3.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - involves a small loss of funds or affects a core functionality of the protocol.
- **Low** - encompasses any unexpected behavior that is non-critical.

3.2 Likelihood

- **High** - a direct attack vector; the cost is relatively low compared to the potential loss of funds.
- **Medium** - only a conditionally incentivized attack vector, with a moderate likelihood.
- **Low** - involves too many or unlikely assumptions; offers little to no incentive.

3.3 Actions required by severity level

- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

4 Executive summary

Hunter Security was engaged by Ouroboros to review their ORX Minter smart contracts.

Overview

Project Name	Ouroboros, ORX Minter
Repository	https://github.com/Ouroboros-Protocol/minter-audit
Commit hash	a158fbf8ca8f01fbdf84e8c6e8afa8168a4371de
Resolution	a6e936a63129daabc42e01fd6109f3ae0cc1aeb7
Methods	Manual review & testing

Scope

contracts/orx/minter/FeeTokenMinter.sol
contracts/orx/minter/ORX.sol
contracts/orx/staking/FeeTokenStaking.sol
contracts/incentives/BackstopPoolIncentives.sol
contracts/incentives/StableLpIncentives.sol

Issues Found

High risk	0
Medium risk	0
Low risk	7
Informational	1

Live Match

FeeTokenMinter	0x4C93D6380D22C44850Bdfa569Df5dD96e278622B	Match
ORX	0xd536E7A9543Cf9867a580B45CEC7F748a1FE11eC	Match
FeeTokenStaking	0xE293DFD4720308c048B63AfE885F5971E135Eb1e	Match
BackstopPoolIncentives	0x91804513f407aaD860968F59A4a8bdE12E71b9b1	Match
StableLpIncentives	0x429E4593Ef49477894a694f332B0d6515d066A55	Match

5 Findings

5.1 Low

5.1.1 User can set themselves as their own referrer claiming additional rewards

Severity: Low

Context: FeeTokenMinter.sol

Description: The function *_appendVestingEntry()* does not validate that the account and referrer addresses are not the same. This allows a user to use their own address while depositing tokens, leading to them earning 10% more tokens.

Recommendation: This is marked as low severity since a user can just use a different wallet address under their control as the referrer. Validation or logical changes should be implemented if this is unwanted behavior.

Resolution: Resolved.

5.1.2 twapDurationMins can be set to a low value

Severity: Low

Context: FeeTokenMinter.sol

Description: The function *setTwapDurationMins* allows the admin to update the *twapDurationMins* variable, which is used for fetching a quote through UniswapV3 Oracle. This variable influences the interval the oracle will look over to determine the time weighed average price. Currently, these are the constraints of the oracle:

```
require(min > 0 && min <= 60);
```

Where *min* is the new value. This means the admin can set this variable to equal 1 minute. Which is not ideal, and might allow for price manipulation attacks because it is a very short interval.

Recommendation: The variable is set to 15 minutes originally in the storage. This is considered a reasonable time interval. It is recommended to have this value be at least 15, as it is now, or increase it for a more stable TWAP quote.

Resolution: Resolved.

5.1.3 Missing balance check to guaranteeing the reward amount

Severity: Low

Context: StableLpIncentives.sol

Description: The *_notifyRewardAmount()* function only works if the *_reward* value passed to it is *50_000_000e18*. The original LQTY code contained an *assert(_reward == lqtyToken.balanceOf(address(this)));* check to confirm the reward amount exists as the balance of the contract. However, this check is omitted in the *StableLpIncentives* contract so there is no guarantee of the reward amount.

Recommendation: Consider implementing the check.

Resolution: Resolved.

5.1.4 Precision loss can lead to no vesting emission

Severity: Low

Context: FeeTokenMinter.sol

Description: The function *calculateReturn()* calculates the amount of *feeTokens* the user will get over their full vesting duration. In one of the three branches, when the deposited amount does not reach the *TOTAL_DEPOSITS_AT_FIXED_RATE* threshold, the following logic is executed:

```
else if ((totalDeposited + tokenIn) <= TOTAL_DEPOSITS_AT_FIXED_RATE) {
    ...
    emittableFeeTokens = tokenIn / FIXED_RATE_FOR_DEPOSIT_TOKEN;
    ...
}
```

If the user supplied a *tokenIn* value that is less than *FIXED_RATE_FOR_DEPOSIT_TOKEN*, then *emittableFeeTokens* will round down to 0. And the user deposit will be vested, but they will get no emissions.

Recommendation: Consider implementing a check that will revert if *emittableFeeTokens* is *0*.

Resolution: Resolved.

5.1.5 Unsafe deposit token transfer

Severity: Low

Context: FeeTokenMinter.sol

Description: The function *buyback()* transfers the *incentiveFee* to the caller through the following code segment:

```
else {
    incentiveFee = (amountIn * incentiveFeeBips) / 10_000;
    amountIn -= incentiveFee;
    depositToken.transfer(msg.sender, incentiveFee);
}
```

Since it uses *transfer* without checking the return value, this transfer can fail, and the execution will continue. The end result is the caller getting their rewards.

Recommendation: Consider using OpenZeppelin's *SafeERC20* library and its *safeTransfer* method.

Resolution: Resolved.

5.1.6 Ownership can be renounced

Severity: Low

Context: FeeTokenMinter.sol

Description: The *renounceOwnership()* function allows the contract owner to give up their owner access. This will leave all functions marked with the *onlyOwner* modifier unusable.

Recommendation: Consider overriding the function and adding a revert statement to it.

Resolution: Resolved.

5.1.7 Lack of zero address checks may lead to broken functionality

Severity: Low

Context: FeeTokenStaking.sol, ORX.sol, BackstopPoolIncentives.sol, StableLpIncentives.sol

Description: The following functions do not validate the addresses passed to them against the zero address before setting the values, allowing for a DoS case for the functionalities that use the affected addresses since *renounceOwnership()* is called after initializing the corresponding parameters:

- *BackstopPoolIncentives.setAddresses()*
- *StableLpIncentives.setParams()*
- *FeeTokenStaking.setAddresses()*
- *ORX.constructor()*
- *ORX.attachMinter()*

Recommendation: Consider validating against the zero address to avoid Denial-of-Service.

Resolution: Resolved.

5.2 Informational

5.2.1 Events not emitted on state change

Severity: Informational

Context: FeeTokenMinter.sol, ORX.sol, FeeTokenStaking.sol, BackstopPoolIncentives.sol

Description: The following functions may emit events for better tracking of contract state changes:

- *FeeTokenStaking.setAddresses()*
- *BackstopPoolIncentives.setAddresses()*
- *FeeTokenMinter.setBuybackIncentiveBips()*
- *FeeTokenMinter.setBuybackCooldownInterval()*
- *FeeTokenMinter.setTwapDurationMins()*
- *FeeTokenMinter.setSlippage()*
- *ORX.attachMinter()*

Recommendation: Consider emitting the aforementioned events.

Resolution: Acknowledged.