code4rena

Gondi

Smart Contract Security Assessment

Version 1.0

Audit dates: Sep 10 — Sep 12, 2024

Audited by: peakbolt
0x1771

Contents

1. Introduction

1.1 About Zenith

Zenith is an offering by Code4rena that provides consultative audits from the very best security researchers in the space. We focus on crafting a tailored security team specifically for the needs of your codebase.

Learn more about us at https://code4rena.com/zenith.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2. Executive Summary

2.1 About Gondi

GONDI is a decentralized peer-to-peer non-custodial NFT lending protocol that aims to offer the most flexible and capital-efficient primitive.

2.2 Scope

Repository	pixeldaogg/token/
Commit Hash	a68f3eed789d11c5760b55259e5400ae04d491ab

2.3 Audit Timeline

DATE	EVENT
Sep 10, 2024	Audit start
Sep 12, 2024	Audit end
Nov 28, 2024	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	2
Medium Risk	0
Low Risk	2
Informational	0
Total Issues	4

3. Findings Summary

ID	DESCRIPTION	STATUS
H-1	Votes for the `veGondiToken` should be zero after lock expiry	Resolved
H-2	Missing update of delegated votes in `extendLock()`	Resolved

L-1	Incorrect event emitted for `transferOwnership()`	Resolved
L-2	`VeGondiToken.tokenURI()` is not compliant with ERC721	Resolved

4. Findings

4.1 High Risk

A total of 2 high risk findings were identified.

[H-1] Votes for the `veGondiToken` should be zero after lock expiry

Severity: High	Status: Resolved

Context:

- VeGondiToken.sol#L127

Description: Users can lock their Gondi token for specific number of weeks in exchange for `veGondiToken`. The votes for each address are accounted in `getVotesPerAddress[]`.

However, `getVotesPerAddress` remains the same even after the veToken lock expires. That means the token owner has no incentive to `extendLock()` when it expires, since the votes are unchanged.

This issue allows the owner to retain the votes even after lock expiry and still be able to burn/redeem the Gondi tokens anytime.

The votes for the veToken should instead be zero when the veToken is no longer locked.

```
    function _getVotingUnits(address account) internal view override
returns (uint256) {
        return getVotesPerAddress[account];
    }
```

Recommendation: This can be fixed by ensuring the delegated votes becomes zero when the veToken lock expires.

Gondi: Fixed with a new checkpoint system to account for new votes and expired votes - commit

Zenith: Verified

[H-2] Missing update of delegated votes in `extendLock()`

Severity: High	Status: Resolved

Context:

- VeGondiToken.sol#L127

Description: `VeGondiToken` inherits the `Votes` abstract contract and utilizes the vote delegation logic for accounting of votes. To ensure the delegated votes are properly accounted, `Votes._transferVotingUnits()` is called during both `mint()` and `burn()`, via the internal function `_update()`.

However, `extendLock()` fails to update the delegated votes as it is not using `transferVotingUnits()` to keep track of the votes.

This issue will cause the delegated votes to be incorrect when an `extendLock()` is performed.

The following test shows that `getVotes(tokenId)` does not match `getVotes(address)` after `extendLock()`.

```
    function testExtendLockIssue() public {
        uint256 lockup = 5;

        vm.startPrank(_user);
        _veGondiToken.delegate(_user);
        uint256 tokenId = _veGondiToken.mint(_user, _amount, lockup);

        console.log("unlockTime == ",
_veGondiToken.getUnlockTime(tokenId));
        console.log("getVotes(tokenId) == ",
_veGondiToken.getVotes(tokenId));
        console.log("getVotes(address) == ",
_veGondiToken.getVotes(_user));

        vm.warp(7 days + 1);
        _veGondiToken.extendLock(tokenId, lockup);
        vm.stopPrank();

        console.log("unlockTime == ",
_veGondiToken.getUnlockTime(tokenId));
        console.log("getVotes(tokenId) == ",
_veGondiToken.getVotes(tokenId));
```

```
        console.log("getVotes(address) == ",
_veGondiToken.getVotes(_user));
    }
```

```
[PASS] testExtendLockIssue() (gas: 382149)
Logs:
  unlockTime ==  5
  Votes(tokenId) ==  5000
  Votes(address) ==  5000
  unlockTime ==  10
  Votes(tokenId) ==  9000
  Votes(address) ==  5000
```

Recommendation: This can be fixed by using `transferVotingUnits()` to update the votes for the account.

Gondi: Fixed with PR-7

Zenith: Verified

4.2 Low Risk

A total of 2 low risk findings were identified.

[L-1] Incorrect event emitted for `transferOwnership()`

Severity: Low	Status: Resolved

Context: TwoStepOwned.sol#L33-L47

Description: `TwoStepOwned.transferOwnership()` emits an event when the ownership is successfully transferred from previous owner to new owner.

As `owner` has been set to `newOwner`, `emit OwnershipTransferred` will not reflect the previous owner in the event.

```
    function transferOwnership(address newOwner) public override
onlyOwner {
        if (pendingOwnerTime + MIN_WAIT_TIME > block.timestamp) {
            revert TooSoonError();
        }
        if (pendingOwner != newOwner) {
            revert InvalidInputError();
        }
        owner = newOwner;
        pendingOwner = address(0);
        pendingOwnerTime = type(uint256).max;

>>>     emit OwnershipTransferred(owner, newOwner);
    }
```

Recommendation: This can be fixed as follows:

```
- emit OwnershipTransferred(owner, newOwner);
+ emit OwnershipTransferred(msg.sender, newOwner);
```

Gondi: Fixed with PR-10

Zenith: Verified

[L-2] `VeGondiToken.tokenURI()` is not compliant with ERC721

Severity: Low	Status: Resolved

Context:

- VeGondiToken.sol#L67-L69

Description:

EIP-721 states that `tokenURI()` should throw an error if `_tokenId` is not a valid NFT.

```
    /// @notice A distinct Uniform Resource Identifier (URI) for a given
asset.
    /// @dev Throws if `_tokenId` is not a valid NFT. URIs are defined in
RFC
    ///  3986. The URI may point to a JSON file that conforms to the
"ERC721
    ///  Metadata JSON Schema".
    function tokenURI(uint256 _tokenId) external view returns (string);
```

However, `VeGondiToken.tokenURI()` fails to revert when an invalid `_tokenId` is provided.

```
    function tokenURI(uint256 _id) public pure override returns (string
memory) {
        return string.concat(URI, Strings.toString(_id));
    }
```

Recommendation: Suggest not to override this and stick to OZ's ERC721 `tokenURI()`, which is compliant with EIP-721 as it will check that the NFT is owned. Refer to ERC721.sol#L88-L93

Gondi: Fixed with PR-5

Zenith: Verified