code4rena

Gondi

Smart Contract Security Assessment

Version 2.0

Audit dates: Oct 21 — Oct 25, 2024

Audited by: peakbolt
spicymeatball

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/florida-contracts
Commit Hash	21e132b8b08731b6fe80563c5c32cbbb63e27350
Mitigation Hash	4804a77049a698ec140ced747a9be5e55a9a3162

2.3 Audit Timeline

DATE	EVENT
Oct 21, 2024	Audit start
Oct 25, 2024	Audit end
Oct 31, 2024	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	3
Medium Risk	1
Low Risk	1
Informational	3
Total Issues	8

3. Findings Summary

ID	DESCRIPTION	STATUS
H-1	ERC20 tokens may become unrecoverable due to incorrect withdrawERC20 implementation	Resolved

H-2	It is possible to steal NFTs from OldERC721Wrapper.sol	Resolved
H-3	Unitialized stash allows anyone to take over it	Resolved
M-1	Incorrect `_PROXY_CODE_SIZE` will truncate init code causing non-compliance to ERC1167	Resolved
L-1	Vault does not support non-standard ERC20 tokens	Acknowledged
I-1	Unused `_emptyData()` function in `ERC1167Factory.sol`	Resolved
I-2	Unused `Owned` inherited contract	Resolved
I-3	Redundant whitelist validation in `_depositERC1155()`	Resolved

4. Findings

4.1 High Risk

A total of 3 high risk findings were identified.

[H-1] ERC20 tokens may become unrecoverable due to incorrect withdrawERC20 implementation

Severity: High	Status: Resolved

Context:

- Stash.sol#L94

Description: Users who deposit their ERC20 tokens in the `Stash.sol` contract may find themselves unable to withdraw their tokens due to the following issue:

```
    function withdrawERC20(
        address tokenAddress,
        uint256 amount
    ) external onlyOwner {
        ERC20 tokenContract = ERC20(tokenAddress);
        uint256 tokenBalance = tokenContract.balanceOf(address(this));

        if (amount > tokenBalance) revert
RequestExceedsAvailableBalance();
>>      tokenContract.transferFrom(address(this), owner, amount);
    }
```

The issue arises because the `transferFrom` function is used instead of the `transfer` function. By default, `transferFrom` checks the allowance of the caller, even if the `from` address is the same as the caller.

Solmate

```
    function transferFrom(
        address from,
        address to,
        uint256 amount
    ) public virtual returns (bool) {
```

```
        uint256 allowed = allowance[from][msg.sender]; // Saves gas for
limited approvals.

>>      if (allowed != type(uint256).max) allowance[from][msg.sender] =
allowed - amount;
```

OpenZeppelin

```
    function transferFrom(address from, address to, uint256 value) public
virtual returns (bool) {
        address spender = _msgSender();
>>      _spendAllowance(from, spender, value);
        _transfer(from, to, value);
        return true;
    }
```

Since `allowance[address(this)][address(this)] = 0`, the transfer will revert.

Recommendation: It is recommended to use the `transfer` function, ideally with the SafeERC20 library.

Gondi: Fixed in PR-427

Zenith: Verified

[H-2] It is possible to steal NFTs from OldERC721Wrapper.sol

Severity: High	Status: Resolved

Context:

- OldERC721Wrapper.sol#L84
- OldERC721Wrapper.sol#L95

Description: Anyone can call the `unwrap` or `unwrapBatch` functions to get ERC721 tokens from the contract:

```
    function unwrap(uint256 tokenId) external {
>>      _burn(tokenId);
        wrapped.transfer(msg.sender, tokenId);
    }
```

This is possible because the Solmate ERC721 implementation doesn't check whether the caller is the owner of `tokenId` in the `_burn` function:

```
    function _burn(uint256 id) internal virtual {
        address owner = _ownerOf[id];
        require(owner != address(0), "NOT_MINTED");
        // Ownership check above ensures no underflow.
        unchecked {
            _balanceOf[owner]--;
        }
        delete _ownerOf[id];
        delete getApproved[id];
        emit Transfer(owner, address(0), id);
    }
```

Recommendation: It is recommended to verify that the caller is either the actual owner or an approved operator of the `tokenId`.

Gondi: Fixed in PR-428

Zenith: Verified.

[H-3] Unitialized stash allows anyone to take over it

Severity: High	Status: Resolved

Context:

- ERC1167Factory.sol#L40-L58

Description: `_deploy()` is missing a call to `initialize(_owner)` to initialize the deployed stash and its owner.

Without the call, anyone can call it and set the stash owner to himself and take over it.

```
function _deploy(
    address implementation,
    bytes32 salt,
    bytes memory data
) internal returns (address proxy) {
    bytes32 m = _initCode();
    assembly {
        // Create the proxy.
        proxy := create2(0, m, _PROXY_CODE_SIZE, salt)
        // Revert if the creation fails.
        if iszero(proxy) {
            mstore(0x00, _DEPLOYMENT_FAILED_ERROR_SELECTOR)
            revert(0x1c, 0x04)
        }

        // Emit the {Deployed} event.
        log3(0, 0, _DEPLOYED_EVENT_SIGNATURE, proxy, implementation)
    }
}
```

Recommendation: Call the `initialize(_owner)` by performing an external call to deployed stash using `data`.

```
function _deploy(
    address implementation,
    bytes32 salt,
    bytes memory data
) internal returns (address proxy) {
    bytes32 m = _initCode();
    assembly {
```

```
            // Create the proxy.
            proxy := create2(0, m, _PROXY_CODE_SIZE, salt)
            // Revert if the creation fails.
            if iszero(proxy) {
                mstore(0x00, _DEPLOYMENT_FAILED_ERROR_SELECTOR)
                revert(0x1c, 0x04)
            }

            // Emit the {Deployed} event.
            log3(0, 0, _DEPLOYED_EVENT_SIGNATURE, proxy, implementation)
        }

+       (bool success, ) = proxy.call(data);
+       if(!success) revert();
    }
```

Gondi: Fixed in [PR-430]

Zenith: Verified

4.2 Medium Risk

A total of 1 medium risk findings were identified.

[M-1] Incorrect `_PROXY_CODE_SIZE` will truncate init code causing non-compliance to ERC1167

Severity: Medium	Status: Resolved

Context:

- ERC1167Factory.sol#L33

Description: The `_PROXY_CODE_SIZE` denote the length of the proxy's bytecode size and is used for the `create2()` opcode to deploy the proxy contract. However, it should be `55` instead of the current `45`.

The truncation will cause an incorrect predicted address that is not compliant to ERC1167, as the init code is different.

The issue was not caught by the tests as the truncated proxy code was responsible for returning the result of the proxy call. And it just so happen that functions in `Stash.sol` like `wrap()` do not have return values.

```
    uint256 internal constant _PROXY_CODE_SIZE = 45;

    function _deploy(
        address implementation,
        bytes32 salt,
        bytes memory data
    ) internal returns (address proxy) {
        bytes32 m = _initCode();
        assembly {
            // Create the proxy.
            proxy := create2(0, m, _PROXY_CODE_SIZE, salt)
            // Revert if the creation fails.
            if iszero(proxy) {
                mstore(0x00, _DEPLOYMENT_FAILED_ERROR_SELECTOR)
                revert(0x1c, 0x04)
            }

            // Emit the {Deployed} event.
            log3(0, 0, _DEPLOYED_EVENT_SIGNATURE, proxy, implementation)
        }
```

```
    }
```

If you see the code below, the first mstore is actually 20 bytes and not 10 bytes.

```
    function _initCode() internal view returns (bytes32 m) {
        bytes20 implementation = bytes20(address(this));

        assembly {
            m := mload(0x40) // 0x40 - 0x5f (32 bytes): currently
allocated memory size (aka. free memory pointer)
            // We want to set deployed bytecode to
0x363d3d373d3d3d363d73{address}5af43d82803e903d91602b57fd5bf3 in that
memory location
            //
            // forgefmt: disable-start

            //@audit this is actually 20 bytes. Also the next mstore uses
an offset of m+0x14, which is 20 bytes.
            mstore(
                m,
>>>
0x3d602d80600a3d3981f3363d3d373d3d3d363d730000000000000000000000000000
            ) // 10 bytes

            mstore(add(m, 0x14), implementation) // 20 bytes

            mstore(
                add(m, 0x28),

0x5af43d82803e903d91602b57fd5bf3000000000000000000000000000000000000
            ) // 15 bytes

            // total initCode size is 10 + 20 + 15 = 45
            // forgefmt: disable-end
        }
    }
```

Recommendation: To resolve this, it is recommended to use OpenZeppelin's `Clones.cloneDeterministic()` as it is battle-tested as compared to writing one from scratch.

Gondi: Fixed in PR-429

Zenith: Verified - by using OpenZeppelin's `Clones.cloneDeterministic()`.

4.3 Low Risk

A total of 1 low risk findings were identified.

[L-1] Vault does not support non-standard ERC20 tokens

Severity: Low	Status: Acknowledged

Context:

- UserVault.sol#L372-L385

Description: As the `_depositERC20()` tracks the deposited amount based on the transferred `_amount`, it will support non-standard ERC20 tokens like rebasing and fee-on-transfer tokens. That is because the amount that can be withdrawn could be lesser (or even more) than what is recorded in the deposit.

```
    function _depositERC20(
        address _depositor,
        uint256 _vaultId,
        address _token,
        uint256 _amount
    ) private {
        if (!_currencyManager.isWhitelisted(_token)) {
            revert CurrencyNotWhitelistedError();
        }
        ERC20(_token).safeTransferFrom(_depositor, address(this),
_amount);

        _vaultERC20s[_token][_vaultId] += _amount;
        emit ERC20Deposited(_vaultId, _token, _amount);
    }
```

Recommendation: This can be acknowledged if the intention is to only support standard ERC20 tokens that do not change balance over time or on certain actions like transfer.

Gondi: Updated documentation at PR-433

Zenith: Acknowledged with documentation updated.

4.4 Informational

A total of 3 informational findings were identified.

[I-1] Unused `_emptyData()` function in `ERC1167Factory.sol`

Severity: Informational	Status: Resolved

Context:

- ERC1167Factory.sol#L120-L124

Description:

Within `ERC1167Factory.sol`, the function `_emptyData()` is unused and can be removed.

```
function _emptyData() internal pure returns (bytes calldata data) {
    assembly {
        data.length := 0
    }
}
```

Recommendation: Remove the unused `_emptyData()` function.

Gondi: Fixed in PR-434

Zenith: Verified

[I-2] Unused `Owned` inherited contract

Severity: Informational	Status: Resolved

Context:

- UserVault.sol#L22
- UserVault.sol#L9

Description: The `UserVault` contract inherits from `Owned`, which is unused as there are no `onlyOwner` modifier in place.

Recommendation: This can be resolved as follows:

```
contract UserVault is
    ERC721,
    ERC721TokenReceiver,
    ERC1155TokenReceiver,
    IUserVault,
-    Owned
{
    ...
    constructor(
        address currencyManager,
        address collectionManager
-    ) ERC721("GONDI_USER_VAULT", "GUV") Owned(tx.origin) {
+    ) ERC721("GONDI_USER_VAULT", "GUV") {
        _currencyManager = AddressManager(currencyManager);
        _collectionManager = AddressManager(collectionManager);
    }
```

Gondi: Fixed in PR-431

Zenith: Verified

[I-3] Redundant whitelist validation in `_depositERC1155()`

Severity: Informational	Status: Resolved

Context:

- UserVault.sol#L394-L396

Description: `_depositERC1155()` has a redundant white list validation on `_token`, which is actually performed in the external functions `depositERC1155()` and `depositERC1155s()`.

Recommendation: This issue can be fixed as follows:

```
    function _depositERC1155(
        address _depositor,
        uint256 _vaultId,
        address _token,
        uint256 _tokenId,
        uint256 _amount
    ) private {
-       if (!_collectionManager.isWhitelisted(_token)) {
-           revert CurrencyNotWhitelistedError();
-       }
        ERC1155(_token).safeTransferFrom(
            _depositor,
            address(this),
            _tokenId,
            _amount,
            ""
        );

        _vaultERC1155s[_token][_tokenId][_vaultId] += _amount;
        emit ERC1155Deposited(_vaultId, _token, _tokenId, _amount);
    }
```

Gondi: Fixed in PR-432

Zenith: Verified