CoinFabrik

Security Audit Report

Crucible

October 2024

V202409

Executive Summary

CoinFabrik was asked to audit the contracts for the **Crucible** project.

During this audit we found one high issue and several minor issues. Also, several enhancements were proposed.

All the issues were resolved. One enhancement was implemented and another was partially implemented.

Scope

The audited files are from the git repositories located at https://github.com/CrucibleNetworksLtd/openmeta-token.git (`openmeta`) on commit `4d40e0bd9844d575010b16cdd86656e791e0a131` and at https://github.com/CrucibleNetworksLtd/StakingContract.git (`staking`) on commit `0cd249e8394a46fa2a4ca2cacc521d6079ae2160.`

The scope for this audit includes and is limited to the following files:

- `openmeta` repository:
  - `src/contracts/ChildERC20.sol`: `TOMetaV3` token implementation. The contract is named `ChildERC20`.
  - `src/contracts/MerkleProof.sol`: Utility to verify merkle trees, based on the one shipped in OpenZeppelin's library.
  - `src/contracts/OpenMetaToken.sol`: OMETA token implementation. The contract is named `OpenMetaToken`. This token has the additional functionality of handling voting, inheriting `OpenZeppelin's` `ERC20Votes` utility contract.
- `staking` repository:
  - `src/contracts/NFTStaking.sol`: contract that can hold ERC721 tokens for some time.

No other files in this repository were audited. Its dependencies are assumed to work according to their documentation. Also, no tests were reviewed for this audit. It must be noted that the `src/contracts/GameItem.sol` file is outside of the scope of this audit. The development team informed us that this contract will not be used in production.

Fixes were checked on commits `b3d2a1ab5a153b68ffb6a81703d7c8caa1890b33` (`openmeta`) and `a84b91bb5ab7ac13809ba3b001d363c3f75c97ce` (`staking`).

Findings

In the following table we summarize the security issues we found in this audit. The severity classification criteria and the status meaning are explained below. This table does not include the enhancements we suggest to implement, which are described in a specific section after the security issues.

Each severity label is detailed in the Severity Classification section. Additionally, the statuses are explained in the Issues Status section.

Id	Title	Severity	Status
HI-01	Merkle Tree Verify Problems	🟧 High	**Resolved**
MI-01	Old Floating Pragma on openmeta	🟨 Minor	**Resolved**
MI-02	Multiple Staking Period Removal	🟨 Minor	**Resolved**
MI-03	Possible Reentrancy Attack	🟨 Minor	**Resolved**
MI-04	Owner Can Take All Rewards	🟨 Minor	**Resolved**

Critical Severity Issues

No issues found.

High Severity Issues

HI-01 Merkle Tree Verify Problems

Repository

- `openmeta`

Location

- `src/contracts/MerkleProof.sol: 22-46`
- `src/contracts/ChildERC20.sol: 49`
- `src/contracts/OpenMetaToken.sol: 55`

Classification

- CWE-1240: Use of a Cryptographic Primitive with a Risky Implementation[1]

Description

The verify function in `src/contracts/MerkleProof.sol` is a modified version of the merkle proof verify function implemented in OpenZeppelin's library, where an `index` is calculated reflecting the path traversed in the proof verification. But the development team did not provide proof that this change is correct, and it is a bad practice to reimplement or modify cryptographic primitives. If this implementation is wrong it may lead to an attacker claiming tokens that do not correspond to them and/or blocking other users from doing their legitimate claims in both the `OpenMetaToken` and `ChildERC20` contracts.

Also take into account that currently there is a race condition when setting a new merkle tree root. If a claim for some funds is in both the old and new merkle tree it can be claimed twice by claiming both before and after the merkle root is set.

Recommendation

Use the `MerkleProof.verify` function provided by `OpenZeppelin` as it is. The leaf passed to it should include both the amount to be claimed and a nonce to avoid replay attacks. Hashing the data to be protected may be necessary. This change would also allow for these merkle trees to share information for claims while avoiding awarding the same claim twice, as it would have the same nonce in several different merkle trees.

Status

Resolved. Now OpenZeppelin's `MerkleProof` is used. To avoid replay attacks and the race condition mentioned in the issue, a `hasClaimed` mapping was introduced to check that a single account cannot claim the rewards twice in both `ChildERC20` and `OpenMetaToken`.

Medium Severity Issues

No issues found.

[1] https://cwe.mitre.org/data/definitions/1240.html

Minor Severity Issues

MI-01 Old Floating Pragma on openmeta

Repository

- `openmeta`

Description

Contracts should be deployed with the same compiler version that they have been thoroughly tested with, and kept up to date with the latest releases of solidity. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that negatively affect the contract system.

Recommendation

Lock the pragma version, replacing `pragma solidity ^0.8.3;` with a specific patch, preferring the most updated version. For example, `pragma solidity 0.8.28`.

Status

Resolved. The solidity version is now pinned to 0.8.28.

MI-02 Multiple Staking Period Removal

Repository

- `staking`

Location

- `src/contracts/NFTStaking.sol: 89-93, 96-104`

Classification

- CWE-436: Interpretation Conflict[2]

[2] https://cwe.mitre.org/data/definitions/436.html

Description

If the same period is added twice by the owner of the `NFTStaking` contract via the `addStakingPeriod` function, it needs to be removed twice via the `removeStakingPeriodFunction` in order to stop being an accepted period.

This may lead to confusion, leading to assume that a period is not valid anymore while it is valid.

Recommendation

Valid periods should be stored in a mapping instead of an array. This has the additional benefit of making the functions `removeStakingPeriod`, `lockNFT` and `extendStake` to take `O(1)` instead of `O(n)` time to look for the period.

Status

Resolved. It was resolved according to the recommendation.

MI-03 Possible Reentrancy Attack

Repository

- `staking`

Location

- `src/contracts/NFTStaking.sol: 36, 39, 45`

Classification

- CWE-841: Improper Enforcement of Behavioral Workflow[3]

Description

The `lockNFT` function in the `NFTStaking` contract does not respect the checks-effects-interaction pattern nor has a reentrancy guard.

The statements in lines 36

```
require(nftCollection.ownerOf(tokenId) == msg.sender, "You don't own this NFT");
```

and 39

[3] https://cwe.mitre.org/data/definitions/841.html

```
nftCollection.transferFrom(msg.sender, address(this), tokenId);
```

invokes an external contract and then the statement in line 45

```
stakedNFTs[msg.sender][tokenId] = StakeInfo(true, unlockTime);
```

changes the state of the contract.

This issues' severity has been reduced because:

1. The `nftCollection` contract is set when constructing the contract and cannot be changed.
2. Most of the NFT collections only make an external call to the recipient of the transfer[4] in line 39 and the recipient is our contract, so potential attack cannot be executed with a vanilla NFT collection.

Recommendation

1. Remove the call in line 36. This check is just wrong as it would not allow an approved address to lock the NFT token and makes two calls to the contract making it impossible to follow the checks-effects-interaction pattern.
2. Move the call in line 39 to the end of the function.

Status

Resolved. It was resolved according to the recommendation.

MI-04 Owner Can Take All Rewards

Repository

- `openmeta`

Location

- `src/contracts/ChildERC20.sol: 63-66, 68-70`

Description

Unlike in the `OpenMetaToken` contract, the owner of the `ChildERC20` contract can set the claim period end date via the `setClaimPeriodEndDate` function and then immediately transfer away all

[4] See `onERC721Received` in https://eips.ethereum.org/EIPS/eip-721.

the rewards via the `sweep` function. So, if the owner account is compromised by an attacker all the claimed tokens may be stolen.

Recommendation

Follow the design decisions made in the OpenMetaToken regarding token claims in the `ChildERC20` contract.

Status

Resolved. The owner cannot change the `claimPeriodEnds` anymore.

Enhancements

These items do not represent a security risk. They are best practices that we suggest implementing.

Id	Title	Status
EN-01	Failing Tests	**Partially implemented**
EN-02	Incorrect Documentation	**Implemented**
EN-03	Pause Mechanism	Not implemented

EN-01 Failing Tests

Repository

- `openmeta`

Description

Running `yarn test` fails the only implemented test in `openmeta`. Having a proper test suite is important for any smart contract project, but it is even more important for the `OpenMetaToken` contract, as its convoluted inheritance tree makes it almost impossible for a human to check if the voting behaves as expected.

Recommendation

1. All tests should pass all the time.

2. Add tests with the proper coverage for all the contracts. This includes both the happy path and the possible reverts in the code.
3. Take special care to test that the votes awarded to each account in the `OpenMetaToken` contract are what is expected.

Status

Partially implemented. Tests pass now but no tests were added to check that votes are handled correctly.

EN-02 Incorrect Documentation

Repository

- `openmeta`

Location

- `src/contracts/ChildERC20.sol: 12-17`
- `src/contracts/OpenMetaToken.sol: 12-17`

Description

Documentation for the `ChildERC20` and `OpenMetaToken` contracts is incorrect.

In both contracts it is claimed that "at creation time the tokens that should be available for the airdrop are transferred to the token contract address" but there is no code in the contract's constructor to do that.

Also, for the `ChildERC20` token, it is claimed that it has "Support for the owner (the DAO) to mint new tokens, at up to 2% PA" but nowhere in the contract code this 2% is enforced.

Recommendation

Change the contracts code and/or documentation to not contradict each other.

Status

Implemented. Wrong documentation was removed.

EN-03 Pause Mechanism

Description

None of the analyzed contracts has any means to stop its operation if a security issue is found.

Recommendation

Consider allowing a privileged role, such as the owner of the contract, to stop its operation to have a mechanism to stop a future attack.

Status

Not implemented.

Other Considerations

The considerations stated in this section are not right or wrong. We do not suggest any action to fix them. But we consider that they may be of interest to other stakeholders of the project, including users of the audited contracts, token holders or project investors.

Upgrades

There are no provisions to upgrade any of the analyzed contracts.

Privileged Roles

These are the privileged roles that we identified on each of the audited contracts.

ChildERC20

Owner

The account with this role can:

1. Transfer away the tokens of any user after the claim period ends via the `sweep` function.
2. Set the claim period end via the `setClaimPeriodEndDate` function.[5]
3. Set the merkle root used to validate token claims via the `setMerkleRoot` function.

[5] This functionallity is not present on commit `b3d2a1ab5a153b68ffb6a81703d7c8caa1890b33`.

This role is managed by inheriting from the `OpenZeppelin`'s `Ownable` utility contract.

The initial owner is the deployer of the contract.

Default Admin

An account with this role can add or remove accounts to roles handled via the `OpenZeppelin`'s `AccessControl` utility contract, including the depositor role and the default admin role. It must be noted that the owner is handled by a different mechanism.

At deployment the only account with this role is the deployer of the contract.

Depositor

An account with this role can mint new tokens via the `deposit` function.

This role is a role managed by inheriting from the `OpenZeppelin`'s `AccessControl` utility contract. The default admin may add or remove accounts with these privileges.

At deployment the only account with this role is `0xb5505a6d998549090530911180f38aC5130101c6`[6].

OpenMetaToken

Owner

The account with this role can:

1. transfer funds away from any account after the claim period ends via the `sweep` function.
2. set, only once, the merkle root used to validate token claims via the `setMerkleRoot` function.
3. mint tokens via the `mint` function.

This role is managed by inheriting from the `OpenZeppelin`'s `Ownable` utility contract.

The initial owner is the `0x9bf13CD856eaCCA29331909A600902F3908E54a7` account[7].

[6] This is hardcoded in the contract's code.
[7] This was hardcoded in the contract's code in commit `4d40e0bd9844d575010b16cdd86656e791e0a131`. In the `b3d2a1ab5a153b68ffb6a81703d7c8caa1890b33` commit the initial owner is the deployer of the contract.

GameItem

Owner

The account with this role can execute the `OwnableFunction` function, which does not have any relevant effect.

This role is managed by inheriting from the `OpenZeppelin`'s `Ownable` utility contract.

The initial owner is the deployer of the contract.

NFTStaking

Owner

The account with this role can add and remove valid staking periods via the `addStakingPeriod` and `removeStakingPeriod` functions.

This role is managed by inheriting from the `OpenZeppelin`'s `Ownable` utility contract.

The initial owner is the deployer of the contract.

Funds Flow Analysis

ChildERC20

The tokens implemented in the contract are transferred in the following functions:

1. `claimTokens`
2. `sweep`

They are minted in the `deposit` function and burned in the `withdraw` function.

No other tokens, nor `ETH`, are used in the code of the contract.

Besides, all the functionality available to manage `ERC20` tokens, inherited from the OpenZeppelin's `ERC20` and `ERC20Permit` contracts, is available[8].

It must be noted that the funds available for the `claimTokens` and `sweep` functions, the development team informed us that they are going to be minted by calling `deposit` via Polygons interface.

[8] For the `b3d2a1ab5a153b68ffb6a81703d7c8caa1890b33` commit `ERC20Votes` was added as well.

OpenMetaToken

The tokens implemented in the contract are transferred in the following functions:

1. `claimTokens`
2. `sweep`

They are minted in the `mint` function and no special provisions for burning the tokens, besides the standard functionality provided by the OpenZeppelin library, are provided.

Besides, all the functionality available to manage `ERC20` tokens, inherited from the OpenZeppelin's `ERC20` and `ERC20Permit` contracts, is available.

It must be noted that the funds available for the `claimTokens` and `sweep` functions, the development team informed us that they are going to be minted by the owner.

No other tokens, nor `ETH`, are used in the code of the contract.

NFTStaking

This is the only analyzed contract that uses a token instead of implementing one.

It interacts with the `nftCollection` set in the constructor.

NFTs are transferred to the contract in the `lockNFT` function and transferred away in the `unlockNFT` function.

About CoinFabrik

CoinFabrik is a research and development company specialized in Web3, with a strong background in cybersecurity. Founded in 2014, we have worked on over 500 decentralization projects, including EVM-based and other platforms like Solana, Algorand, and Polkadot. Beyond development, we offer security audits through a dedicated in-house team of senior cybersecurity professionals, working on code in languages such as Substrate, Solidity, Clarity, Rust, TEAL, and Stellar Soroban.

Our team has an academic background in computer science, software engineering, and mathematics, with accomplishments including academic publications, patents turned into products, and conference presentations. We actively research in collaboration with universities worldwide, such as Cornell, UCLA, and École Polytechnique in Paris, and maintain an ongoing collaboration on knowledge transfer and open-source projects with the University of Buenos

Aires, Argentina. Our management and people experience team has extensive expertise in the field.

Methodology

CoinFabrik was provided with the source code, including automated tests that define the expected behavior. Our auditors spent one week auditing the source code provided, which includes understanding the context of use, analyzing the boundaries of the expected behavior of each contract and function, understanding the implementation by the development team (including dependencies beyond the scope to be audited) and identifying possible situations in which the code allows the caller to reach a state that exposes some vulnerability. Without being limited to them, the audit process included the following analyses.

- Arithmetic errors
- Outdated version of Solidity compiler
- Race conditions
- Reentrancy attacks
- Misuse of block timestamps
- Denial of service attacks
- Excessive gas usage
- Missing or misused function qualifiers
- Needlessly complex code and contract interactions
- Poor or nonexistent error handling
- Insufficient validation of the input parameters
- Incorrect handling of cryptographic signatures
- Centralization and upgradeability

After delivering a report with our findings, the development team had the opportunity to comment on every finding and fix the issues they considered convenient. Once fixed and/or commented, our team ran a second review process to verify that the changes to the code effectively solve the issues found and do not unintentionally add new ones. This report includes the final status after the second review.

Severity Classification

Security risks are classified as follows:

🟥 Critical	These are issues that we manage to exploit. They compromise the system seriously. Blocking bugs are also included in this category. They must be fixed **immediately**.
🟧 High	These refer to a vulnerability that, if exploited, could have a substantial impact, but requires a more extensive setup or effort compared to critical issues. These pose a significant risk and **demand immediate attention**.
🟨 Medium	These are potentially exploitable issues. Even though we did not manage to exploit them or their impact is not clear, they might represent a security risk in the near future. We suggest fixing them **as soon as possible**.
🟨 Minor	These issues represent problems that are relatively small or difficult to take advantage of, but might be exploited in combination with other issues. These kinds of issues do not block deployments in production environments. They should be taken into account and be fixed **when possible**

Issue Status

An issue detected by this audit has one of the following statuses:

- **Unresolved:** The issue has not been resolved.
- **Resolved:** Adjusted program implementation to eliminate the risk.
- **Partially Resolved:** Adjusted program implementation to eliminate part of the risk. The other part remains in the code, but is a result of an intentional decision.
- **Acknowledged:** The issue remains in the code, but is a result of an intentional decision. The reported risk is accepted by the development team.
- **Mitigated:** Implemented actions to minimize the impact or likelihood of the risk.

Disclaimer

This audit report has been conducted on a **best-effort basis within a tight deadline defined by time and budget constraints**. We reviewed only the specific smart contract code provided by the client at the time of the audit, detailed in the Scope section. We do not review other components that are part of the solution: neither implementation, nor general design, nor business ideas that motivate them.

While we have employed the latest tools, techniques, and methodologies to identify potential vulnerabilities, **this report does not guarantee the absolute security of the contracts, as**

undiscovered vulnerabilities may still exist. Our findings and recommendations are suggestions to enhance security and functionality and are not obligations for the client to implement.

The results of this audit are valid solely for the code and configurations reviewed, and any modifications made after the audit are outside the scope of our responsibility. CoinFabrik disclaims all liability for any damages, losses, or legal consequences resulting from the use or misuse of the smart contracts, including those arising from undiscovered vulnerabilities or changes made to the codebase after the audit.

This report is intended exclusively for the **Crucible project** and should not be relied upon by any third party without the explicit consent of CoinFabrik. Blockchain technology and smart contracts are inherently experimental and involve significant risk; users and investors should fully understand these risks before deploying or interacting with the audited contracts.

Changelog

Date	Description
2024-10-29	Initial report based on commits `4d40e0bd9844d575010b16cdd86656e791e0a131` and `0cd249e8394a46fa2a4ca2cacc521d6079ae2160`.
2024-11-05	Checks fixes on commits `b3d2a1ab5a153b68ffb6a81703d7c8caa1890b33` and `a84b91bb5ab7ac13809ba3b001d363c3f75c97ce`. Move the `GameItem.sol` file out of the scope of this audit.