CoinFabrik

Security Audit Report

Yala

February 2025

V202411

Executive Summary

CoinFabrik was asked to audit the contracts for the Yala project.

This report, as requested by the client, only includes some files in the `yala-contract-core` repository.

During the initial phase of this audit we found one critical issue and several low issues. Also, several enhancements were proposed. Of those issues, the critical one and one of the low ones were resolved. The rest is still unresolved. None of the enhancement proposals were implemented.

A new critical issue was introduced during the fixes of the initial issues.

Scope

While the original audit had a broader scope, this report only includes the following files in the https://github.com/yalaorg/yala-contract-core repository:

- `contracts/NFT/YetiNFT.sol`: it contains the `Yetiasd`[1] contract, an ERC721 token with facilities to give tokens away.
- `contracts/NFT/stake.sol`: It contains the `Stake` contract, it implements an ERC721 token that represents a staked NFT.

The audit was conducted on commit `00f262174f7bc81ac2218cc12f80a878a2b1ada9`. Fixes were checked on commit `3f351d1ee4376cd52a6ebb676be25be677742f3f`.

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.

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.

[1] The `YetiAsd` contract was renamed as `Yeti` for commit `3f351d1ee4376cd52a6ebb676be25be677742f3f`.

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

Id	Title	Severity	Status
CR-01	Can Unstake After Transferring Away	🟥 Critical	**Resolved**
CR-02	Yeti NFT Freeze	🟥 Critical	Unresolved
LO-01	No URI for NFTs	🟨 Low	**Resolved**
LO-02	Single NFT Staking Bypass	🟨 Low	Unresolved
LO-03	Floating Pragma on core	🟨 Low	Unresolved

Critical Severity Issues

CR-01 Can Unstake After Transferring Away

Location

- `contracts/NFT/stake.sol`

Found on Commit

- `00f262174f7bc81ac2218cc12f80a878a2b1ada9`

Classification

- CWE-863: Incorrect Authorization[2]

Description

If a user makes a stake in the `Stake` contract via the `stake` function and then transfers the resulting NFT away, they still can unstake this NFT and obtain the underlying staked NFT.

The recipient of the transferred staking NFT cannot unstake it, making the generated staking NFT effectively useless.

[2] https://cwe.mitre.org/data/definitions/863.html

Recommendation

Either allow the owner, and only the owner, of the staking NFT to unstake it or make the `Stake` contract to not be a NFT contract.

Status

Resolved. The `Stake` contract is not an ERC721 token anymore. Checked on commit `3f351d1ee4376cd52a6ebb676be25be677742f3f`.

CR-02 Yeti NFT Freeze

Location

- `contracts/NFT/YetiNFT.sol: 16, 82-88, 113-118, 123-127`

Found on Commit

- `3f351d1ee4376cd52a6ebb676be25be677742f3f`

Description

The mapping `OWNEDNFTS` of the `Yeti` contract contains an array that increases in length for each NFT that is transferred to a user or minted to it. The array is then iterated each time the `burn` function is executed or when the `_beforeTokenTransfer` private function is executed, that is invoked in both `safeTransferFrom` functions and the `transferFrom` function.

So, when a new NFT is awarded to a user it increases the gas cost to do transfers and burns and if the number of NFTs of a single user is high enough the gas cost may exceed the maximum allowed for a transactions, effectively making the user unable to burn its tokens, transfer them away or both.

Recommendation

In order to implement the functionality of the Yeti contract there is no need to store an array of NFT ids for each user and iterate it. Use just mappings instead.

Status

Unresolved.

High Severity Issues

No issues found.

Medium Severity Issues

No issues found.

Low Severity Issues

LO-01 No URI for NFTs

Location

- `contracts/NFT/YetiNFT.sol:27`
- `contracts/NFT/stake.sol:40`

Found on Commit

- `00f262174f7bc81ac2218cc12f80a878a2b1ada9`

Description

In the `Yetiasd` contract, the base URI is defined as `""`, and in the `Stake` contract it is defined as `"adjsjfhdjfhs"`[3]. Neither generates a valid and absolute URI for the NFTs. This may lead to problems when showing those NFTs in marketplaces.

Recommendation

Use a proper base URI for all the implemented NFTs.

Status

Resolved. The Yeti contract[4] now has `"https://assets.yala.org/json/"` as the base URI and the Stake contract is not an ERC721 token anymore. Checked on commit `3f351d1ee4376cd52a6ebb676be25be677742f3f`.

[3] This is the actual string in the codebase.
[4] `YetiAsd` in the initial commit.

LO-02 Single NFT Staking Bypass

Location

- `contracts/NFT/stake.sol:21`

Found on Commit

- `00f262174f7bc81ac2218cc12f80a878a2b1ada9`

Description

The `stake` function of the `Stake` contract checks that the invoking account does not have a staked NFT. But this check is not effective because making new accounts is trivial in EVM blockchains, allowing any user to have as many accounts as they want. This allows any user to stake as many NFTs as they choose.

Recommendation

Allow any account to have many staked NFTs.

Status

Unresolved.

LO-03 Floating Pragma on core

Found on Commit

- `00f262174f7bc81ac2218cc12f80a878a2b1ada9`

Location

- `contracts/NFT/stake.sol:2`
- `contracts/NFT/YetiNFT.sol:2`

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 the `pragma solidity` statements that allow for multiple solidity compilers to be used with a specific patch, preferring the most updated version. For example, `pragma solidity 0.8.28`. Also it is required to thoroughly test all the changed contracts not to introduce additional bugs.

Status

Unresolved.

Enhancements

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

Id	Title	Status
EN-01	No Limits or Rewards in Stake Contract	Not implemented
EN-02	Import Ownable2Step	Not implemented
EN-03	Simplify NFT Eligibility	Not implemented

EN-01 No Limits or Rewards in Stake Contract

Location

- `contracts/NFT/stake.sol`

Found on Commit

- `00f262174f7bc81ac2218cc12f80a878a2b1ada9`

Description

The staking mechanism implemented in the `Stake` contract does not have any rewards for staking the NFT nor any limitations on when the NFT can be unstaken.

Recommendation

Add some reasonable limitations and rewards to make the staking useful.

Status

Not implemented.

EN-02 Import Ownable2Step

Location

- `contracts/NFT/YetiNFT.sol`

Found on Commit

- `00f262174f7bc81ac2218cc12f80a878a2b1ada9`

Description

The `YetiAsd` contract includes a custom two-step process for transferring ownership. The token's implementation could be simplified by incorporating OpenZeppelin's `Ownable2Step` and leveraging its functionality through inheritance.

Recommendation

Use `Ownable2Step` instead of reimplementing it.

Status

Not implemented.

EN-03 Simplify NFT Eligibility

Location

- `contracts/NFT/YetiNFT.sol`

Found on Commit

- `00f262174f7bc81ac2218cc12f80a878a2b1ada9`

Description

Currently, the `setEligible` function manages user eligibility through an `ELIGIBLE` mapping, which flags users who can claim NFTs. This design involves additional storage operations, consuming more gas when updating mappings for each eligible user and requiring explicit eligibility management through two separate mappings – `ELIGIBLE` and `NFT`.

By incorporating a zero-value token ID check directly within the `setEligible` function, the contract can simplify eligibility management by relying solely on the `NFT` mapping. If `NFT[user]` returns a non-zero token ID, the user is deemed eligible to claim their `NFT`.

Recommendation

Add a zero-check to the `setEligible` function and remove the `ELIGIBLE` mapping.

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 mechanisms to upgrade in the analyzed contracts.

Privileged Roles

These are the privileged roles that we identified on each of the audited contracts that are not taken from the MakerDAO project.

Yetiasd

Owner

The owner of the contract can:

1. add users that can claim NFTs via the `setEligible` function.
2. mint expired NFTs to itself via the `recycleNFT` function.
3. nominate a new owner via the `nominateNewOwnership` function. In order to effectively have a new owner, the nominated account must accept the ownership by calling the `acceptOwnership` function.

The initial owner of the contract is the deployer of the contract.

Stake

There are no privileged roles in this contract.

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, and general documentation about the project. Our auditors spent twelve weeks 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

Fixes were checked on the scope that was reduced by the client.

Severity Classification

Security risks are classified as follows[5]:

🟥 Critical	• Manipulation of governance voting result deviating from voted outcome and resulting in a direct change from intended effect of original results • Direct theft of any user funds, whether at-rest or in-motion, other than unclaimed yield • Direct theft of any user NFTs, whether at-rest or in-motion, other than unclaimed royalties • Permanent freezing of funds • Permanent freezing of NFTs • Unauthorized minting of NFTs • Predictable or manipulable RNG that results in abuse of the principal or NFT • Unintended alteration of what the NFT represents (e.g. token URI, payload, artistic content) • Protocol insolvency

[5] This classification is based on the smart contract Immunefi severity classification system version 2.3. https://immunefi.com/immunefi-vulnerability-severity-classification-system-v2-3/

High	• Theft of unclaimed yield • Theft of unclaimed royalties • Permanent freezing of unclaimed yield • Permanent freezing of unclaimed royalties • Temporary freezing of funds • Temporary freezing NFTs
Medium	• Smart contract unable to operate due to lack of token funds • Block stuffing • Griefing (e.g. no profit motive for an attacker, but damage to the users or the protocol) • Theft of gas • Unbounded gas consumption • Security best practices not followed
Low	• Contract fails to deliver promised returns, but doesn't lose value • Other security issues with minor impact

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 Yala team 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
2025-02-07	Initial Report
2025-03-07	Check fixes in reduced scope and report new critical issue