SOLIDIFIED

Audit Report for TinyTap CourseNFT - October 20, 2022

Summary

Audit Report prepared by Solidified covering the Course NFT smart contract for TinyTap.

Process and Delivery

Independent Solidified experts performed an unbiased and isolated audit of the code below. The final debrief took place on September 23, 2022, and the results are presented here.

Audited Files

The source code has been supplied in a Github Gist:

https://gist.github.com/NegruGeorge/2442719549b3580c9797088bda413575/dffe26efc415293b2340c587a64200f641ee71a3

Update: The TinyTap team provided a new version with fixes to issues encountered and updates on September 27, 2022 in the following repository:

https://github.com/tinytap/smart-contracts
Commit number: `b9e5a63d88c23aec22006e59019ec7875b17eb8e`

Update: The TinyTap team provided a new version with fixes to issues encountered and updates on October 5, 2022 in the following repository:

https://github.com/tinytap/smart-contracts
Commit number: `47de46a4999206c1779f6d2eea87e0d96c5fc0e1`

Intended Behavior

The contract implements Course NFTs for TinyTap.

Findings

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

Note, that high complexity or lower test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

Criteria	Status	Comment
Code complexity	Low	-
Code readability and clarity	High	-
Level of Documentation	High	-
Test Coverage	High	-

Issues Found

Solidified found that the TinyTap CourseNFT contract contains 0 critical issues, 0 major issues, 2 minor issues, and 2 informational notes.

We recommend issues are amended, while informational notes are up to the team's discretion, as they refer to best practices.

Issue #	Description	Severity	Status
1	The setGeneralReceiver function does not change the default royalty receiver	Minor	Resolved
2	Change of royalty fees can be set unreasonably high post-minting	Minor	Acknowledged
3	The owner of the contract is set as the royalty receiver if the creator is not set	Note	Resolved
4	Miscellaneous Comments	Note	Resolved / Acknowledged

Critical Issues

No critical issues have been found.

Major Issues

No major issues have been found.

Minor Issues

1. The `setGeneralReceiver` function does not change the default royalty receiver

The `CourseNFT` constructor calls the `setGeneralReceiver` function and provides the `address receiver`. However, this function does nothing more than setting the value of `generalReceiver` and is otherwise unused. If `setGeneralReceiver` is subsequently called to change the value of the `generalReceiver`, the default royalty `receiver` remains set to the previous receiver unless `setDefaultRoyalty` is explicitly called with the new address.

Recommendation
We recommend using only the `setDefaultRoyalty` function to manage the default royalty receiver and the royalty fee and removing the `setGeneralReceiver` function as well as the `generalReceiver` storage variable.

Status
Resolved

2. Change of royalty fees can be set unreasonably high post-minting

The EIP-2981 standard allows contracts to signal a royalty fee paid to the creator of the NFT every time the NFT is sold or re-sold. This royalty fee is specified as a percentage of the sale price in basis points as a fraction of `10_000`. However, the owner of the `CourseNFT` contract can change the royalty fee of an already minted NFT at any time by using the `setTokenRoyalty` function. This can be used to increase the royalty fee to a very high percentage, resulting in an unexpected loss of value for the NFT holder.

Recommendation

We recommend adding a reasonable maximum value for the royalty fee `feeNumerator` or only allowing the royalty fee of an already minted NFT to be reduced over time.

Status

Pending

Informational Notes

3. The owner of the contract is set as the royalty receiver if the creator is not set

During minting, if the `feeNumerator` is not 0, and there is no `_creator` specified, the royalty receiver is set as the owner of the contract instead of the default royalty receiver.

Recommendation

We recommend using the default `receiver` address returned by the `ERC2981.royaltyInfo` function if the contract owner is not intended to be the royalty receiver.

Status
Resolved

4. Miscellaneous Comments

The following are suggestions to improve the overall code quality and readability.

- Remove commented code: `L21`, `L22` (resolved)
- Unnecessary import: Openzeppelin's `ReentrancyGuard` (resolved)
- The comment for the `walletOfOwner` function might be misleading (resolved)
- In the function `walletOfOwner`, the parameter `_owner` shadows the state variable `Ownable._owner` (resolved)
- Consider emitting events for important state variable changes (acknowledged)

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of TinyTap or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Oak Security GmbH