SOLIDIFIED

Audit Report for Cool Cats Shadow Wolves - April 17, 2023

Summary

Audit Report prepared by Solidified covering the Cool Cats Shadow Wolves contracts.

Process and Delivery

iIndependent Solidified experts performed an unbiased and isolated audit of the code below. The final debrief took place on April 17, 2023, and the results are presented here.

Audited Files

The source code has been supplied in the following source code repository:

Repo: https://github.com/coolcatsnft/shadow-wolves-contract/tree/audit
Commit hash: `a256228ab53a79235412d92be3347c035614af35`

```
contracts/
├── ShadowWolves.sol
└── minting
    └── IntoTheFracture.sol
```

Intended Behavior

The audited code base implements the Shadow Wolves NFT and a minting contract to burn Fractures for Shadow Wolves.

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	-

SOLIDIFIED

Audit Report for Cool Cats Shadow Wolves - April 17, 2023

Issues Found

Solidified found that the Cool Cats Shadow Wolves contracts contain no critical issues, no major issue, 1 minor issue, 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 same merkle proof can be consumed multiple times	Minor	Resolved
2	Burn window is not necessarily one consecutive window	Note	Acknowledged
3	Merkle root can be unset with allowListEnabled	Note	Resolved

Critical Issues

No critical issues have been found.

Major Issues

No major issues have been found.

Minor Issues

1. The same merkle proof can be consumed multiple times

The function `hasValidMerkleProof` does not mark a proof that was used as consumed and a valid merkle proof can be consumed multiple times. Because of that, one malicious address can circumvent this gating mechanism completely. If this address is an EOA, it can call the `enterFracture` function for other users (as `msg.sender` needs to have a valid merkle proof, not the owner of the NFT). If a malicious contract is added to the tree, this could even be automated with a function that allows anyone to call `enterFracture` over this contract, no matter if the address is contained in the tree or not.

Recommendation

Consider one of the following options:

- Requiring that a valid merkle proof is provided for each owner (instead of for `msg.sender` only) is provided. In that case, it would not be a problem that each proof can be consumed multiple times, as the described circumvention method no longer would work.
- Only allowing a proof to be used once (or for a configurable amount of time).

Informational Notes

2. Burn window is not necessarily one consecutive window

There is an administrative function `setBurnWindow` to change the start and end of the burn window. This function can be called at any time, for instance also during the burn window or after it has ended. Because of that, there is not necessarily one consecutive burn window, as the owner can start a second one after the first one has ended or extend it indefinitely.

Recommendation
Consider restricting the setBurnWindow if it should be guaranteed that there is only one consecutive window.

3. Merkle root can be unset with allowListEnabled

Currently a `merkleRoot` can be set to anything in `setMerkleRoot` regardless of the smart contract state. If this `merkleRoot` is set to 0 while `allowlistEnabled` is true, `hasValidMerkleProof` can be bypassed.

Recommendation
Add logic to allow setting a `merkleRoot` to 0 only when the `allowlistEnabled` variable is set to false in `setMerkleRoot` similar to the logic in `setAllowlistEnabled` when a `merkleRoot` is set to 0.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Cool Cats 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