SOLIDIFIED

Audit Report for Fraktal - October 28, 2021

Summary

Audit Report prepared by Solidified covering the Fraktal smart contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The final debrief took place on October 28, 2021, and the results are presented here.

Audited Files

The source code has been supplied in a public source code repository:

https://github.com/FraktalNFT/contracts

Commit number: `28ab4e7ce24a3525c48c633ea32ba4d7a87e5b29`

UPDATE: Fixes received on November 8, 2021
Final commit number: `da8d95e65b85858cde3f045224ca41809132111a`

File List:

```
├── FraktalFactory.sol
├── FraktalMarket.sol
├── FraktalNFT.sol
├── IFraktalNFT.sol
├── IPaymentSplitter.sol
├── PaymentSplitterUpgradeable.sol
└── TransparentUpgradeableProxy.sol
```

Intended Behavior

Fraktal is a DAO powering a fractional NFT marketplace.

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	Medium	-
Code readability and clarity	Medium	-
Level of Documentation	Low	-
Test Coverage	Medium	-

SOLIDIFIED

Audit Report for Fraktal - October 28, 2021

Issues Found

Solidified found that the Fraktal contracts contain no critical issues, no major issues, 3 minor issues, and 4 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	FraktalMarket.sol: Function withdrawAccruedFees() can potentially fail when transferring ETH to a smart contract	Minor	Resolved
2	FraktalMarket.sol: Function setFee() does not enforce an upper limit for _newFee	Minor	Resolved
3	FraktalNFT.sol: Function setMajority() does not enforce an upper limit for newValue	Minor	Resolved
4	FraktalNFT, FraktalMarket, PaymentSplitterUpgradeable: external Visibility should be preferred	Note	Resolved
5	Violation of Checks-Effects-Interactions pattern	Note	Resolved
6	Wide Solidity compiler target	Note	Resolved
7	Miscellaneous Notes	Note	-

Critical Issues

No critical issues have been found.

Major Issues

No major issues have been found.

Minor Issues

1. FraktalMarket.sol: Function withdrawAccruedFees() can potentially fail when transferring ETH to a smart contract

Function `withdrawAccruedFees()` calls `transfer()` when sending ETH to `_msgSender`, which only forwards 2300 gas. In cases where the `_msgSender` address is a smart contract whose fallback function consumes more than 2300 gas, the call will always fail. This will have the side effect of potentially preventing smart contracts (e.g. DAOs) from receiving transfers.

For a more in-depth discussion of issues with `transfer()` and smart contracts, please refer to `https://diligence.consensys.net/blog/2019/09/stop-using-soliditys-transfer-now/`

Recommendation
Replace instances of `transfer()` with `call()`. Alternatively, `AddressUpgradeable.sendValue()` can also be used instead (this is already being used correctly in `PaymentSplitterUpgradeable.release()`).

Note 1

In case the chosen fix is replacing `transfer()` with `call()`, make sure that `rescueEth()` is not vulnerable to reentrancy attacks (as it currently would be).

Note 2

The same issue also exists in the following functions: `FraktalMarket.rescueEth()`, `FraktalMarket.makeOffer()`, and `FraktalNFT.createRevenuePayment()`.

Status

Resolved

2. FraktalMarket.sol: Function setFee() does not enforce an upper limit for _newFee

---

Since `fee` is a percentile amount, it would make sense for the newly set `_newFee` to have an upper limit that does not exceed 100%.

Recommendation

Set an upper limit for `_newFee` that does not exceed 100%.

Status

Resolved

3. FraktalNFT.sol: Function setMajority() does not enforce an upper limit for newValue

---

Recommendation

Set an upper limit for `newValue` that does not exceed `10000`.

Status

Resolved

Informational Notes

---

4. FraktalNFT, FraktalMarket, PaymentSplitterUpgradeable: external Visibility should be preferred

---

Recommendation

Consider using `external` visibility for functions that are only supposed to be called from outside the contract.

Status

Resolved

5. Violation of Checks-Effects-Interactions pattern

The contract `FraktalMarket` violates the checks-effects-interaction pattern in several occurrences. The problem is not too serious here because of the limited gas as part of `transfer`, but it is still recommended to use the pattern.

Recommendation
The pattern makes sure that you don't call an external function until you've done all the internal work you need to do. Implementation details can be found here http://solidity.readthedocs.io/en/develop/security-considerations.html#re-entrancy.

Status
Resolved

6. Wide Solidity compiler target

The contracts use different compiler versions defined by pragmas. It is considered best practice to stick to a single compiler version throughout the codebase.

Recommendation
Choose a single compiler version.

Status
Resolved

7. Miscellaneous Notes

- FraktalMarket.sol: `>=0` check in function `setFee()` is redundant since `uint16` is never negative.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Fraktal Technologies 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.

Solidified Technologies Inc.