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`

File List:

```
├── Fraktal1155.sol
├── 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, 4 major issues, 6 minor issues, and 5 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	Fraktal1155.sol: Missing approval validation	Major	Pending
2	Fraktal1155.sol: Batch transfer method can be used to bypass the validation	Major	Pending
3	Fraktal1155.sol: Locked funds - fee collected	Major	Pending
4	Fraktal1155.sol: Non-standard parameter order for function safeTransferFrom()	Major	Pending
5	FraktalMarket.sol: Function withdrawAccruedFees() can potentially fail when transferring ETH to a smart contract	Minor	Pending
6	Fraktal1155.sol: Function buy() will fail if the user sends the exact totalPrice amount	Minor	Pending
7	Fraktal1155.sol: Function buy() does not refund any extra ETH sent	Minor	Pending
8	FraktalMarket.sol: Function setFee() does not enforce an upper limit for _newFee	Minor	Pending
9	FraktalNFT.sol: Function setMajority() does not enforce an upper limit for newValue	Minor	Pending
10	Fraktal1155.sol: Avoid using tx.origin	Minor	Pending
11	Fraktal1155.sol: Function rescueEth() inconsistently uses both _msgSender() and msg.sender	Note	-

12	FraktalNFT, FraktalMarket, PaymentSplitterUpgradeable: external Visibility should be preferred	Note	-
13	Violation of Checks-Effects-Interactions pattern	Note	-
14	Wide Solidity compiler target	Note	-
15	Miscellaneous Notes	Note	-

Critical Issues

No critical issues have been found.

Major Issues

1. Fraktal1155.sol: Missing approval validation

The function `safeTransferFrom()` is defined as public, but is missing the validations required to ensure that the caller has enough permissions. TransferFrom method should ensure that the from address has approved the caller to transfer funds on behalf - which is missing in this function implementation. This will allow any user to call the method directly and withdraw funds.

Recommendation

Consider adding the validation to ensure the caller has enough permission to use transfer from.

2. Fraktal1155.sol: Batch transfer method can be used to bypass the validation

The function `safeTransferFrom()` uses the input tokenId to perform extra validations. The `ERC1155` standard allows batch transfer methods but the contract does not override the `safeBatchTransferFrom()` to include these validations. This will allow the caller to skip these validations.

Recommendation

Consider using the `_beforeTokenTransfer` hook to add validations that apply to all transfers.

3. Fraktal1155.sol: Locked funds - fee collected

The function `buy()` collects the fee from the user but there is no way to withdraw the fee in the contract. This will cause the funds to be locked in the contract forever.

```
127|   uint256 totalPrice = (listing.price * _numberOfShares) + fee;
```

Furthermore, the contract allows the buyer to send more funds than required which can never be withdrawn.

Recommendation
Consider adding a method to allow the admin to withdraw the fee. Return any funds that are not used to the user.

4. Fraktal1155.sol: Non-standard parameter order for function safeTransferFrom()

The function `safeTransferFrom()` defines the '`to`' parameter before the '`from`' parameter, which makes it not conform to the `ERC1155` standard.

Recommendation
Fix the parameter order to match the `ECR1155` standard.

Minor Issues

5. 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()`, `FraktalNFT.createRevenuePayment()` and `Contract.rescueEth()`.

6. Fraktal1155.sol: Function buy() will fail if the user sends the exact totalPrice amount

---

Function `buy()` requires that the sent ETH be greater than `totalPrice`, instead of greater than or equal to.

Recommendation

Require that `msg.value >= totalPrice`.

7. Fraktal1155.sol: Function buy() does not refund any extra ETH sent

---

Function `buy()` does not refund any extra ETH sent back to the buyer.

Recommendation

Refund any amount that exceeds `msg.value - totalPrice`.

Note

The same issue also exists for functions `FraktalMarket.buyFraktions()` and `FraktalMarket.makeOffer()`.

8. 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%.

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

---

Recommendation

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

10. Fraktal1155.sol: Avoid using tx.origin

The contract uses `tx.origin` in several places to retrieve the caller address. This approach usually comes with it's own risks. An attacker can scam the user to call this method through a proxy.

Recommendation

Consider using a different approach to retrieve the caller account. For example, the user address can be sent as a parameter value after restricting the function call to a limited set of callers.

Informational Notes

11. Fraktal1155.sol: Function rescueEth() inconsistently uses both _msgSender() and msg.sender

Function `rescueEth()` inconsistently mixes and matches between `_msgSender()` and `msg.sender`. This can potentially create vulnerabilities if `_msgSender()` gets overridden later on and the two values are not identical anymore (e.g. if a gas network is paying for the transaction fees).

Recommendation
Consider replacing `msg.sender` with `_msgSender()`.

12. 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.

13. Violation of Checks-Effects-Interactions pattern

The contracts `FraktalMarket` and `Fraktal1155` in several places violate the checks-effects-interaction pattern. 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.

14. 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.

15. Miscellaneous Notes

1. Fraktal1155.sol: The contract named `Contract` shadows a reserved word. Consider renaming the contract.
2. Fraktal1155.sol: `uint256` is always greater than or equal to 0. Any require statements checks this are invalid and can be removed.
3. FraktalMarket.sol: `>=0` check in function `setFee()` is redundant since `uint16` is never negative.

Audit Report for Fraktal - October 28, 2021

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.