SOLIDIFIED

Audit Report for Quidd Inc. - May 31, 2022

Summary

Audit Report prepared by Solidified covering the Quidd mintable smart contracts.

Process and Delivery

Independent Solidified experts performed an unbiased and isolated audit of the code. The debrief was on 24 May 2022.

Audited Files

The source code has been supplied in the form of a source code repository:

https://github.com/Quidd/quidd-mintables-poc

Final Commit hash: `6dfdf3fe35f532b595c34111b3d559c6a1573ab8`

The scope was reduced to the following files:

```
contracts
├── QuiddMintables.sol
├── QuiddMintablesBase.sol
├── roles
│   ├── AdminRole.sol
│   ├── MinterRole.sol
│   └── PauserRole.sol
├── royalties
│   └── ERC2981
│       ├── ERC2981TokenIDMask.sol
│       └── IERC2981.sol
└── token_id
    ├── QuiddTokenIDv0.sol
    └── TokenIDMaskRestrictor.sol
```

Intended Behavior

The smart contracts implement an ERC-721 token with permissioned minting and royalty payments. **Royalty payments are optional and up to the frontend to honor, they are not enforced by the smart contracts, which only inform of the appropriate value and destination.**

SOLIDIFIED

Audit Report for Quidd Inc. - May 31, 2022

Code Complexity and Test Coverage

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases have their 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 equate to a higher risk. Certain bugs are more easily detected in unit testing than a security audit and vice versa. It is, therefore, more likely that undetected issues remain if the test coverage is low or non-existent.

Criteria	Status	Comment
Code complexity	Low	-
Code readability and clarity	Medium-high	There are blocks of commented-out code and test contracts that could be removed from the codebase.
Level of Documentation	Medium	Whilst the codebase is relatively well commented, no additional documentation has been provided and the README file is a leftover from the Hardhat boilerplate project.
Test Coverage	High	-

SOLIDIFIED

Audit Report for Quidd Inc. - May 31, 2022

Issues Found

Solidified found that the Quidd mintable contracts contain no critical issue, no major issue and 2 minor issues, 5 informational notes complete the report.

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

Issue #	Description	Severity	Status
1	Minter role's abilities increase vulnerability to key compromise or misuse	Minor	Resolved
2	ERC2981TokenIDMask.sol: Royalties can be set to 100 percent	Minor	Resolved
3	QuidMintablesBase.sol: Contract exposes dangerous functionality unless overridden	Note	Resolved
4	TokenIDMaskRestrictor.sol: Contract may fail with excessively large numbers of token ID masks	Note	Resolved
5	AdminRole.sol, MinterRole.sol, PauserRole.sol: Unnecessary function implementation already present in parent contract	Note	Resolved
6	QuiddMintablesBase.sol, QuiddMintables.sol: reentrancy risk in composability scenarios	Note	Resolved

SOLIDIFIED

Audit Report for Quidd Inc. - May 31, 2022

Critical Issues

No issues found

Major Issues

No issues found

Minor Issues

1. Minter role's abilities increase vulnerability to key compromise or misuse

The `mint` and `unmint` functions can be executed by the privileged `MinterRole`. This centralized design is compounded by the fact that unminting allows this role to remove and reassign a user's already issued token at will, equivalent to an infinite allowance transferFrom. Apart from intentional abuse scenarios, this also significantly increases the impact of accidental key disclosure, for example through a compromised web service.

Recommendation
Consider separating the minter and unminter functionality into different roles, with keys handled by segregated web services

Update
The team has implemented separated roles for minting and unminting.

2. `ERC2981TokenIDMask.sol`: Royalties can be set to 100 percent

The `_setDefaultRoyalties()` function does not have any checks beyond ensuring that the range is between 0 and 100 percent. This means that royalties could be misused to cover the full value of a transaction. Apart from intentional abuse scenarios, this significantly increases the impact of an admin key compromise, with which an attacker could siphon user's funds, on top of disposing of their Quidd erc721 tokens.

Recommendation
Consider limiting royalties to a reasonable maximum percentage, which would greatly mitigate this type of attack as well as reduce the potential for error

Update
The team has implemented a maximum royalty limit of 20%.

Notes

3. QuidMintablesBase.sol: Contract exposes dangerous functionality unless overridden

The contract is intended to be imported and overridden by subcontracts. However, it is not an abstract contract and could be instantiated on its own. The default `mint` and `unmint` functions have no form of access control and can be called by anyone. Relying on subcontracts to remove dangerous functionality by overriding functions is bad practice, since it is error prone and may lead to code reuse issues.

Recommendation
Consider declaring the contract abstract or making `mint` and `unmint` internal functions.

4. TokenIDMaskRestrictor.sol: Contract may fail with excessively large numbers of token ID masks

The `_tokenIDAllowed` method iterates over a dynamic array of token id masks. Since this function is executed in write transactions it may theoretically be susceptible to failing due to the block gas limit, should the array grow too large.
Whilst this is extremely unlikely in the intended use case, it is considered bad practise to allow for iterations over dynamic data structure without bounds.

Recommendation
Consider placing a limit on the maximum number of masks allowed.

5. AdminRole.sol, MinterRole.sol, PauserRole.sol: Unnecessary function implementation already present in parent contract

The functionality of `renounceRole` is already implemented in the parent contract, with the same access controland is not required, being redundant and more expensive to run than the parent's function.

Recommendation
Consider removing the unnecessary functions to save on deployment and runtime gas costs.

6.QuiddMintablesBase.sol, QuiddMintables.sol: reentrancy risk in composability scenarios

ERC721 contains a couple of reentrant functions. These are clearly differentiated from the standard `transferFrom` and `mint` functions by bearing the "safe" prefix, and developers know how to handle them safely. The mint function exposed by these contracts uses `_safeTransfer` when handling previously unminted tokens, and therefore is reentrant as well, and not protected by any mutex. This break of pattern might go unnoticed to developers building around the Quidd tokens, who might call mint in an unsafe context without realizing they are now open to reentrancy attacks.

Recommendation
Consider adding a warning notice to the mint function natspec comments on both contracts, or introducing a reentrancy guard or other form of mutex.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Quidd Inc. 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 from legal and financial liability.

Oak Security GmbH