QuillAudits

Audit Report
December, 2023

For

Table of Content

Table of Content

Executive Summary

Project Name	Locksonic
Project URL	*https://locksonic.io/*
Overview	Locksonic stands out in the digital asset marketplace with its commitment to making NFT trading more accessible through affordable pricing. This approach ensures that the diverse world of NFTs is open to a broader audience, emphasizing the artistic and collectible value of digital assets.
Audit Scope	*https://github.com/locksonic/xnft-contracts-locksonic/tree/feat/liquidity-pool/contracts*
Contracts in Scope	XNFTClone.sol XNFTFactory.sol XNFTLiquidityPool.sol *interfaces* IXNFTFactory.sol IXNFTClone.sol IXNFTLiquidityPool.sol *exension* XNFTAdmin.sol XNFTAssetManager.sol XNFTBase.sol XNFTMint.sol
Commit Hash	b294ef7572012759bd1cab96c95f01f35d66048d
Language	Solidity
Blockchain	Ethereum
Method	Manual Analysis, Functional Testing, Automated Testing

Review 1	28th November 2023 - 13th November 2023
Updated Code Received	13th December 2023
Review 2	13th December 2023 - 15th December 2023
Fixed In	*https://github.com/locksonic/xnft-contracts-locksonic/tree/fixes/audit*

Number of Security Issues per Severity

8
Issues Found

🟥 High 🟨 Medium

🟩 Low 🟪 Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	**1**	**1**	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	**3**	**1**	**2**

Checked Vulnerabilities

- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ DoS with Block Gas Limit
- ✓ Transaction-Ordering Dependence
- ✓ Use of tx.origin
- ✓ Exception disorder
- ✓ Gasless send
- ✓ Balance equality
- ✓ Byte array
- ✓ Transfer forwards all gas
- ✓ ERC20 API violation
- ✓ Malicious libraries
- ✓ Compiler version not fixed
- ✓ Redundant fallback function
- ✓ Send instead of transfer
- ✓ Style guide violation
- ✓ Unchecked external call
- ✓ Unchecked math
- ✓ Unsafe type inference
- ✓ Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, and their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Hardhat, Foundry.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

A high-severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impacts and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

High Severity Issues

No issues were found.

Medium Severity Issues

1. Changes in account info after minting leads to incorrect results

Path

XNFTAdmin.sol

Function

updateAccount

Description

In the **XNFTAdmin.sol** contract, specifically at line **78**, the **updateAccount** function grants the operator the ability to modify account information. This presents a potential risk of introducing discrepancies as certain variable values can be altered. For instance, the operator could decrease the **_maxMintPerTx** variable, leading to incongruities. This scenario might result in users observing distinct limits for different accounts, causing confusion and potential issues, especially for users who have already minted multiple NFTs under the same account.

Recommendation

We recommended is to permit increases in the **accountInfo** values like **_maxMintPerTx** rather than allowing decreases.

Status

Resolved

2. Unorthodox method used to transfer NFT from user account

Path

XNFTLiquidityPool.sol

Function

_redeem

Description

In the **XNFTLiquidityPool.sol** contract, specifically at line **175**, the _redeem function utilizes a specialized function, **xnftClone.nftRedemption**, to redeem NFT from a user's account without obtaining explicit approval. This practice deviates from industry standards and smart contract best practices, as it poses a potential security risk. It is recommended to adhere to established practices and ensure that explicit user approval is obtained before the **XNFTLiquidityPool** contract takes ownership of the NFT. This adjustment will enhance security and align with the expected behavior in decentralized applications, preventing unauthorized access to users' NFTs.

Recommendation

We recommended to take the explicit approval from the user before redeeming NFT.

LockSonic Team's Comment

One of XNFTs key feature is a built-in liquidity pool which enables the NFT owner to make a direct redemption and claim from the smart contract. Adding an additional approval would increase the gas fees to the users and may provide the user with a sense that he is interacting with an external smart contract which is not part of the integrated system. The use case of an ERC721 approval is when interacting with an external system and not for built-in features.

Status

Acknowledged

3. Lack of zero address check leads to unexpected burning of fee

Path

XNFTAdmin.sol

Function

_accountFeeAddress

Description

In the **XNFTAdmin.sol** contract, there is a potential vulnerability where the operator, providing the **_accountFeeAddress** during the minting of NFTs, can mistakenly set the **_accountFeeAddress** to **0x0**. The current implementation lacks explicit checks to ensure that the provided address is non-zero, allowing the **_accountFeeAddress** to be set to an invalid value. This could lead to fees transferred during minting being inadvertently burned, impacting the expected behavior of the contract.

Recommendation

It is recommended to enhance the code by incorporating checks for non-zero addresses during the assignment of **_accountFeeAddress** to prevent this potential vulnerability.

Status

Resolved

4. Operator's ability to modify account mintPrice promotes unfair primary market

Path

XNFTAdmin.sol

Function

updateAccount

Description

In the **XNFTAdmin.sol** contract, the **updateAccount** function poses a potential concern as it grants the operator the ability to change the mintPrice of the NFT collection at will. This flexibility allows the operator to decrease the NFT **mintPrice**, potentially causing financial losses for users who purchased NFTs at a higher **mintPrice**. Users who bought NFTs before the decrease in mintPrice would receive less during the redemption of their NFTs compared to those who made purchases after the mintPrice reduction. This raises fairness and transparency issues, impacting users who invested in the NFT collection.

Recommendation

It is recommended to not allow to decrease in the **mintPrice** once the current timestamp is greater than the already set **mintTimestamp**.

Status

Resolved

Low Severity Issues

5. Lack of Initialization Sequence in XNFTBase.sol

Path

XNFTBase.sol

Function

__NFTweetsBase_init()

Description

In the **XNFTBase.sol** contract, there is an absence of proper initialization for the inherited contracts. Specifically, the initialization sequence for the **OwnableUpgradeable** contract is not explicitly called within the **__NFTweetsBase_init()** function. This could lead to issues if another contract inherits from **XNFTBase** and misses the initialization of the **OwnableUpgradeable** contract.

Recommendation

It is advised to include explicit calls to initialize the inherited contracts (**OwnableUpgradeable**, **ReentrancyGuardUpgradeable**, **PausableUpgradeable**) within the **__NFTweetsBase_init()** function to ensure proper initialization and avoid potential issues for contracts inheriting from **XNFTBase**.

Status

Resolved

6. Missing royaltyFee deduction during redeem

Path

XNFTLiquidityPool.sol

Function

redeem and **claim**

Description

The **XNFTLiquidityPool.sol** contract exhibits inconsistent behavior in its **redeem** and **claim** functions. The redeem function, which involves sending a given tokenId to the contract, does not deduct a royalty fee, essentially allowing users to sell NFTs without paying royalties. On the other hand, the claim function, which represents a purchase operation, deducts the royalty fee. This inconsistency poses a risk, as it can lead to a lack of fairness in fees between selling and buying operations, impacting users differently depending on the operation they engage in.

Recommendation

To address the inconsistency in royalty fees between the redeem and claim functions in the **XNFTLiquidityPool.sol** contract, it is recommended to implement a uniform fee structure. Both selling (**redeem**) and buying (**claim**) operations should adhere to the same royalty fee mechanism. This adjustment will enhance fairness and consistency in the fee structure, ensuring a more equitable experience for users engaging in both NFT selling and buying transactions.

Status

Acknowledged

Informational Issues

7. Incorrect naming convention

Description

In certain sections of the codebase, there are instances where variables and modifiers have been named inaccurately. For example, in **contracts/extensions/XNFTAdmin.sol** at line 9, the modifier named **OnlyOperator** could cause confusion, as it implies that either **msg.sender** should be an **_operator** or the owner of the contract. To accurately represent its functionality, the modifier's name should be changed to **OnlyOperatorOrOwner**. Similarly, in **contracts/extensions/XNFTBase.sol** at line 18, the variable **maxMintPerTx** may mislead as it seems to denote the maximum number of NFTs minted per transaction, while it is actually used to represent the maximum number of NFTs minted per user. To avoid confusion, it is recommended to rename it appropriately, perhaps to **maxMintPerUser**.

Recommendation

We recommend to rename OnlyOperator to OnlyOperatorOrOwner while rename maxMintPerTx to maxMintPerUser.

Status

Resolved

8. Missing compile-time type check leads to incorrect behavior

Path

XNFTAdmin.sol

Function

abi.encodeWithSelector

Description

In **XNFTAdmin.sol** at **[L#136]** & **[L#151]**, the Current implementation is using **abi.encodeWithSelector** to encode function calls during the creation of a BeaconProxy instance. The use of **abi.encodeWithSelector** lacks compile-time type checks for parameters, which may lead to the inadvertent provision of incorrect parameter types. This could result in runtime transaction reverts if the provided type is not supported. The absence of a type check poses a risk for future upgrades where the type of initialize function parameters may change, but adjustments may not be made at **XNFTAdmin.sol: [L#136]** & **[L#151]**.

Recommendation

We recommend using **abi.encodeCall** instead of **abi.encodeWithSelector** as it provides the compile type check. **abi.encodeCall** is introduced in the recent releases of solidity.

Status

Resolved

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the LockSonic codebase. We performed our audit according to the procedure described above.

Some issues of Medium, Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in LockSonic smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of LockSonic smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the LockSonic to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

850+
Audits Completed

$30B
Secured

$30B
Lines of Code Audited

Follow Our Journey

Audit Report
December, 2023

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com