SQWID

SMART CONTRACT AUDIT

zokyo

June 16th 2022 | v. 1.0

Security Audit Score

PASS

Zokyo's Security Team has concluded that this smart contract has good enough security level to pass security qualifications to be listed on digital asset exchanges.

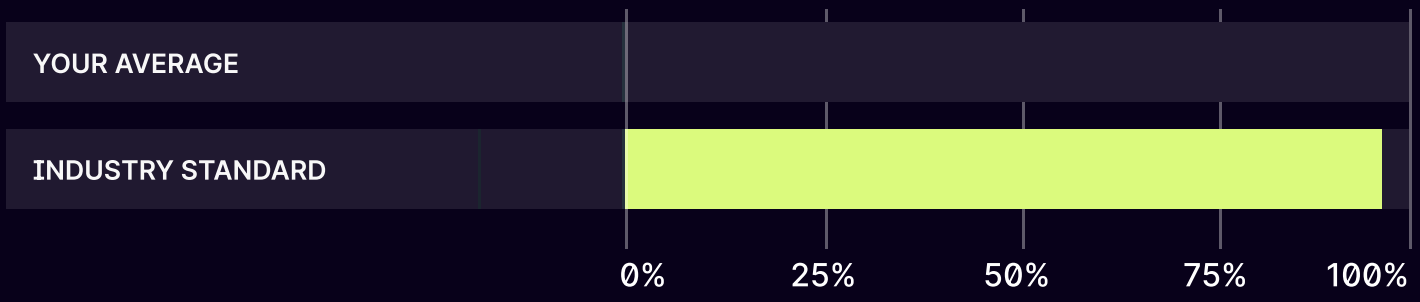
TECHNICAL SUMMARY

This document outlines the overall security of the Sqwid smart contracts, evaluated by Zokyo's Blockchain Security team.

The scope of this audit was to analyze and document the Sqwid smart contract codebase for quality, security, and correctness.

Contract Status

Testable Code



Contracts set contains native unit-tests, though test coverage check was out of the scope for this audit.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we at Zokyo recommend that the Sqwid team put in place a bug bounty program to encourage further and active analysis of the smart contract.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The Smart contract's source code was taken from the Sqwid repository.

Repository: https://github.com/sqwid-app/sqwid-core

Initial commit: 060c5bf02c6736005b3e18aba07cf3b6db0993a3

Last commit: 494c7374acca04d7745e5d38fe87a9ee15c364de

Within the scope of this audit Zokyo auditors have reviewed the following contract(s):

- metarouter_gateway
- metarouter_gateway
- metarouter_gateway
- metarouter_gateway
- metarouter_gateway
- metarouter_gateway
- metarouter_gateway
- metarouter_gateway

Throughout the review process, care was taken to ensure that the contract:

- Implements and adheres to existing standards appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of resources, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Whether the code meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of Sqwid smart contracts. To do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. Part of this work includes writing a unit test suite using the Truffle testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01	Due diligence in assessing the overall code quality of the codebase.
02	Cross-comparison with other, similar smart contracts by industry leaders.
03	Testing contract logic against common and uncommon attack vectors.
04	Thorough, manual review of the codebase, line-by-line.

Executive Summary

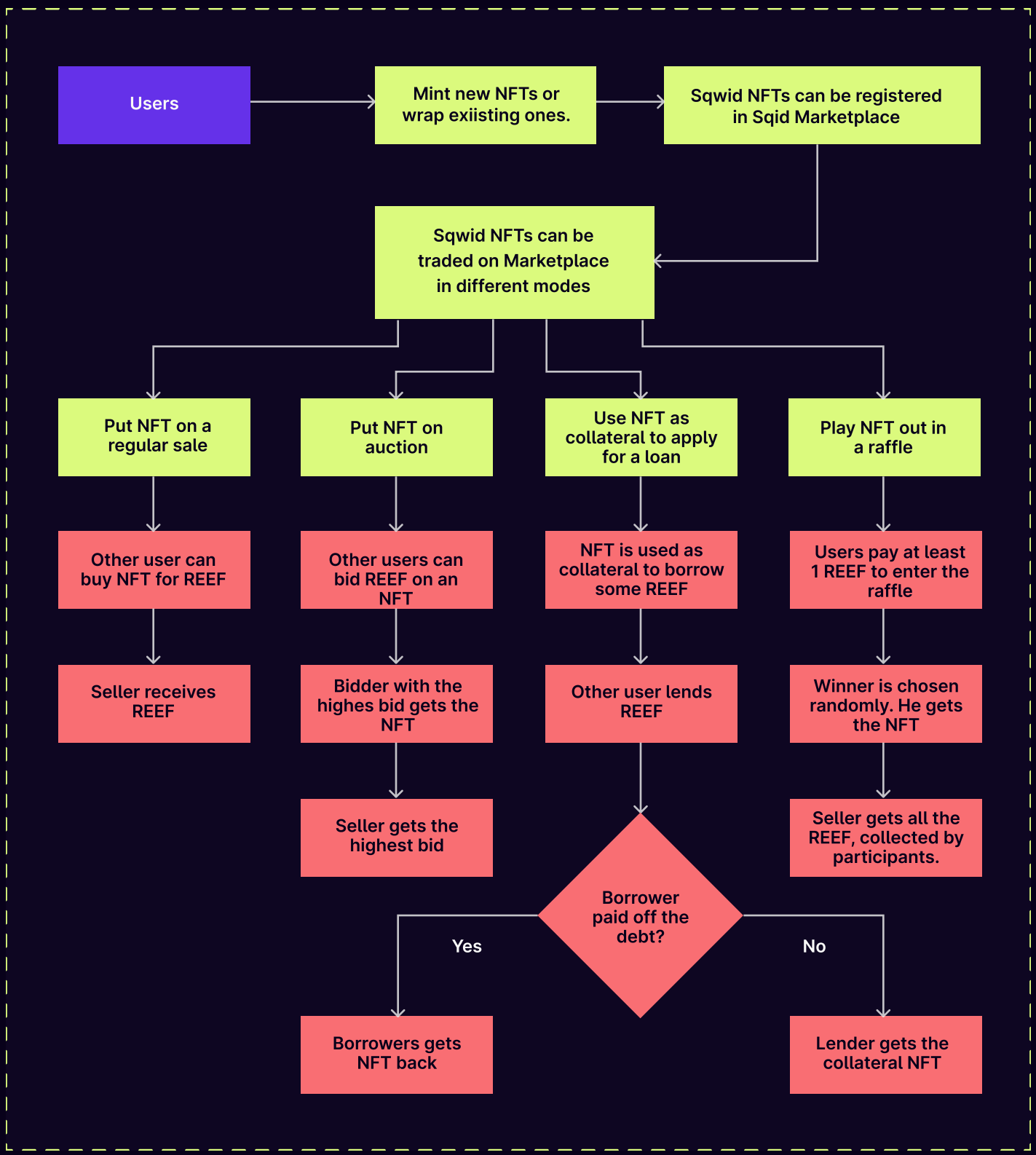
Sqwid protocol is an NFT marketplace with several features. During the audit, Zokyo team has verified both Sqwid NFT contract and NFT Marketplace. Minting and burning functionalities of Sqwid NFT were properly checked, as well as wrapping of external NFTs. NFT Marketplace was verified to correctly operate with Sqwid NFTs. All the functionalities, such as selling, raffle, auction and NFT loan, were properly checked by the Security team.

Several critical issues were found during audit. One of them regards to the ability of miners to affect the results of the raffle. The Sqwid team is acknowledged about the issue, however due to absence of Oracles on Reef chain, issue could not be resolved at the moment. Another critical issue was connected to accuracy loss during entering the raffle, which potentially leads to the loss of REEF tokens. The Sqwid team is acknowledged about this issue as well and it was decided to verify the accuracy on Front-end side of the protocol.

All other issues were successfully fixed by the Sqwid team. The team also performed some gas optimization fixes to decrease gas spendings and ensure that the protocol cannot be blocked due to gas limitations.

Overall, the code is well-structured, most of the issues were resolved by the Sqwid team of developers, however there is still an issue, which should be resolved, once Oracles are available on Reef chain.

PROTOCOL OVERVIEW



STRUCTURE AND ORGANIZATION OF DOCUMENT

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. Issues tagged “Verified” contain unclear or suspicious functionality that either needs explanation from the Customer’s side or it is an issue that the Customer disregards as an issue. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

CRITICAL | UNRESOLVED

Miners are able to manipulate the result of the raffle.

SqwidMarketplace.sol: function endRaffle(), line 751.
In order to determine the winner of the raffle, a pseudo-random number is generated, based on the information of the block. Miners are able to manipulate the transaction and some of the blocks in order to manipulate the outcomes of the raffle.

The issue is described in the following article:
https://betterprogramming.pub/how-to-generate-truly-random-numbers-in-solidity-and-blockchain-9ced6472dbdf

Recommendation:
remove to_string()

From client:
The Sqwid team states, that there are no off-chain Oracles in the Reef chain and opts with current solution vulnerable for miners manipulation.

Not the whole amount of REEF is transferred due to accuracy loss.
SqwidMarketplace.sol: functions enterRaffle(), endRaffle().

When a user enters a raffle, his message value is divided by 1e18, losing the accuracy of the message value, since Solidity doesn't support float numbers. For example, if a user sends `2500000000000000000` (2.5 REEF) and this value is divided by 1e18, the result would be equal 2 (instead of 2.5).
During the execution of endRaffle(), the total value is multiplied by 1e18, giving `2000000000000000000` (2 REEF). This way, some amount of REEF tokens might get stuck on contract.

Recommendation:
Remove the division.

Post audit:
Sqwid team took this approach in order to facilitate the calculation of the pseudorandom winner, given that the value of 1 REEF is currently $0.0035, so Sqwid team has enough margin to discard that value in case of incorrectly sending a fractional part. The official frontend will take this into account and allow only amounts with the accuracy of 1 REEF. Sqwid team however added a @notice in the contract to make this fact explicit, in case the users want to interact directly with the smart contract or other frontends are developed.

Deprecated ETH transfer.

Governance.sol: function withdraw(), line 163.
SqwidMarketplace: withdraw() line 214, fundLoan() line 896, repayLoan() line 918, _createItemTransaction() line 1181
Due to the Istanbul update there were several changes provided to the EVM, which made .transfer() and .send() methods deprecated for the ETH transfer. Thus it is highly recommended to use .call() functionality with mandatory result check, or the built-in functionality of the Address contract from OpenZeppelin library. This should be done in order to mitigate any possible future update of EVM for the Reef Chain.

Recommendation:
Correct ETH sending functionality.

Iteration through the whole storage array.

1. Governance.sol: function _resetProposals(), lines 422, 428, 432.Iteration is performed through several arrays with proposals, including the executed one. In case, there are a lot of elements in arrays, iteration might consume more gas than allowed per transaction, rejecting the whole transaction. Issue is marked as high, since it might prevent protocol from updating list of owners and minimum confirmation value.
2. SqwidERC1155Wrapper.sol: function _getWrappedTokenId(), line 77.Iteration through all wrapped tokens. In case there are a lot of tokens wrapped, function would prevent from wrapping new tokens due gas limit per one transaction.

Recommendation:

1. Consider removing executed proposals from arrays to decrease the number of elements. In case, proposals should not be removed, consider to split the reset process in several transactions to ensure its correct execution.
2. Consider creating additional mapping for linking the token of external contract to wrapped token id instead of iteration.

Recommendation:

1. The Sqwid team has changed the structure of the Governance. Now the mappings are used to track proposals and arrays to store the active ones. Also, a limit of active proposals per owner has been added.
2. The way of linking external tokens with wrapped tokens has been changed as well, from using an array to a mapping.

MEDIUM | RESOLVED

Payable function without interaction with msg.value.

SqwidMarketplace.sol: function mint().
Function is marked as payable, however there is no interaction with msg.value. All REEF, accidentally sent with this function will be locked on contract.

Recommendation:
Either remove the payable modifier or add interaction with msg.value, for example, update of user balance with function _updateBalance().

From-client:
Payable modifier was removed by the team.

LOW | RESOLVED

Check the supply of the token to verify that the token is ERC721.

SqwidERC1155.sol: function unwrapERC721(), line 360.
Currently, the balance of the message sender is checked to be equal 1, in order to verify that the token is ERC721 standard. However it is possible that other tokens from this set would be on other users' balances and the balance of the message sender would be 1.

Recommendation:
Check the supply of the token.

From-client:
According to the Sqwid team, the purpose of the validation was to ensure, user has enough balance of token. However, for the sake of clearness in the error returned, a new validation of the correctness of the token standard has been added in both unwrapERC721 and unwrpERC1155 functions.

INFORMATIONAL | RESOLVED

Validate correctness of proposal before creating.

Governance.sol: functions proposeOwnersChange(), proposeMinConfirmationsChange(). Currently, the correctness of values passed to proposals are verified in functions executeOwnersChange() in lines 323, 328 and executeMinConfirmationsChange() in lines 404. The correctness of a proposal is not verified during its creation, allowing users to flood the system with incorrect proposals and increase the amount of elements in proposals' arrays.

Recommendation:
Validate the correctness of proposal in functions proposeOwnersChange(), proposeMinConfirmationsChange().

From-client:
Validations have been added in the moment of proposal creation.

INFORMATIONAL | RESOLVED

Proposals have no deadline.

Proposals are not limited in timeline, which means that some proposals might be voted for later, when they appear not to be actual for protocol. Such behavior might be potentially used by malicious actors(In case owners are compromised).

Recommendation:
Verify the correctness of current functionality or add a deadline, after which, the proposal cannot be executed.

From-client:
A deadline has been added to all proposals.

INFORMATIONAL | RESOLVED

Validate tokenURI parameter.

SqwidERC1155.sol: functions mint(), mintBatch().
Parameter “tokenURI” in function mint() and values from array parameter “tokenURIs” lack validation that strings are not empty.

Recommendation:
Validate the fact that provided tokenURIs are not empty strings.

INFORMATIONAL | RESOLVED

Unused function.

SqwidERC1155.sol: function _asSingleftonArray().
Private function _asSingleftonArray() is defined, however it is not used across the code.

Recommendation:
Remove unused function.

INFORMATIONAL | RESOLVED

Unused import.

NftMimeTypes.sol: import of ERC165.sol.
Imported contract is neither inherited nor used as interface

Recommendation:
Remove unused imports.

Constant usage is preferred.

NftRoyalties.sol: royaltyInfo(), line 29.
SqwidMarketPlace,sol: fundLoan(), line 892; _createItemTransaction() line 1175
The number `10000` is used as the accuracy divider, thus it is preferable to have it as constant so its usage will be consistent across the contracts
The same applies to royalty value check in NftRoyalties.sol, _setTokenRoyalty(), line 54 - verify that `5000` value should be set as constant or can be changeable by the admin.

Recommendation:
Consider constant usage.

From-client:
The value of the accuracy divider will not be used outside the contract and thus shouldn't be moved to a separate constant.
The max royalty value, however, has been converted into a constant, so that it can be referenced in other contracts.

	Governance.sol	SqwidERC1155.sol	SqwidMarketplace.sol
Re-entrancy	Pass	Pass	Pass
Access Management Hierarchy	Pass	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass	Pass
Unexpected Ether	Pass	Pass	Pass
Delegatecall	Pass	Pass	Pass
Default Public Visibility	Pass	Pass	Pass
Hidden Malicious Code	Pass	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass	Failed
External Contract Referencing	Pass	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass	Pass
Unchecked CALL Return Values	Pass	Pass	Pass
Race Conditions / Front Running	Pass	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass	Pass
Floating Points and Precision	Pass	Pass	Failed
Tx.Origin Authentication	Pass	Pass	Pass
Signatures Replay	Pass	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass	Pass

	NftMimeTypes.sol	NftRoyalties.sol	SqwidERC1155Wrapper.sol
Re-entrancy	Pass	Pass	Pass
Access Management Hierarchy	Pass	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass	Pass
Unexpected Ether	Pass	Pass	Pass
Delegatecall	Pass	Pass	Pass
Default Public Visibility	Pass	Pass	Pass
Hidden Malicious Code	Pass	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass	Pass
External Contract Referencing	Pass	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass	Pass
Unchecked CALL Return Values	Pass	Pass	Pass
Race Conditions / Front Running	Pass	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass	Pass
Floating Points and Precision	Pass	Pass	Pass
Tx.Origin Authentication	Pass	Pass	Pass
Signatures Replay	Pass	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass	Pass

We are grateful to have been given the opportunity to work with the Sqwid team.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

Zokyo's Security Team recommends that the Sqwid team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo