LIMIT BREAK

SMART CONTRACT AUDIT

zokyo

June 12th 2023 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

TECHNICAL SUMMARY

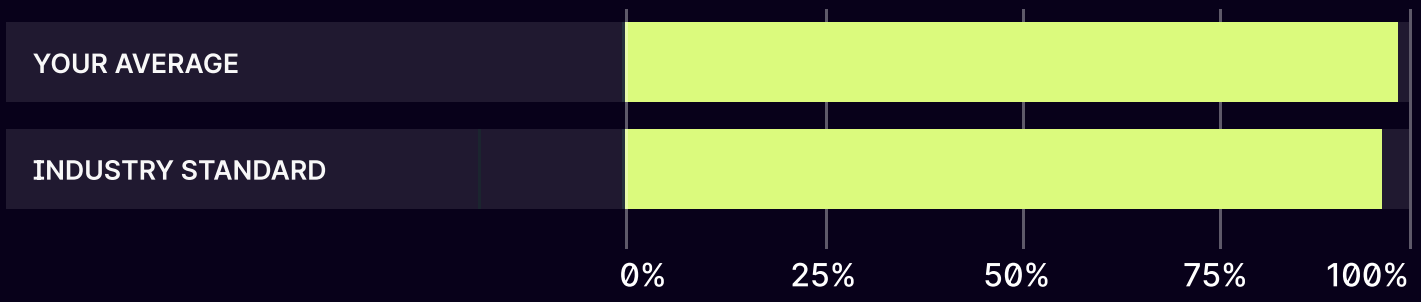
This document outlines the overall security of the Limit Break smart contracts evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the Limit Break smart contracts codebase for quality, security, and correctness.

Contract Status

There was 1 critical issue found during the audit. (See Complete Analysis)

Testable Code



98.8% of the code is testable, which is above the industry standard of 95%.

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

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Limit Break repository:
https://github.com/limitbreakinc/minimum-floor-operator-zokyo

Last commit - `0e29e681b810315859344ab6b9c562fe01af039f`

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- IOwnable.sol
- PaymentProcessor.sol
- IPaymentProcessor.sol
- PaymentProcessorDataTypes.sol

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most resent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Limit Break smart contracts. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. Part of this work includes writing a test suite using the Hardhat 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 contracts logic against common and uncommon attack vectors.
04	Thorough manual review of the codebase line by line.

Executive Summary

During the audit, the Limit Break team successfully resolved one critical issue that was identified. Additionally, two issues of high severity, three issues of medium severity, and some low issues were discovered. A detailed explanation of these findings can be found in the "Complete Analysis" section.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the Limit Break team and the Limit Break team is aware of it, but they have chosen to not solved it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. 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

FINDINGS SUMMARY

#	Title	Risk	Status
1	Edge case in business logic leads to stolen funds	Critical	Resolved
2	Edge case in business logic leads to stolen nft	High	Resolved
3	Using transferFrom can lead to edge cases when the token is a weird implementation of erc20	High	Resolved
4	No sanity checks when setting a security policy to a collection	Medium	Resolved
5	Logic edge case if a chain re-organisation happens	Medium	Acknowledged
6	DOS through sandwich attack on ceilingPrice	Medium	Acknowledged
7	No sanity checks when adding a new coin	Low	Resolved
8	Missing zero address check for account parameter	Low	Resolved

CRITICAL | RESOLVED

Edge case in business logic leads to stolen funds

In contract PaymentProcessor.sol, to identify if the nft token is either ERC1155 or ERC721 the structure MatchOrder have a property named protocol, which is an enum and can have the values TokenProtocols.ERC1155 or TokenProtocols.ERC721, this check is done so that the contact knows which transferFrom function to call, safeTransferFrom in the case of ERC1155 or transferFrom in the case of ERC721, However this can be exploited because that parameter is not part of neither the buyer or the seller signature so it can be manipulated, let's take the following example:

1. Alice (Seller) and Bob(buyer) agree to make an otc deal where Alice sells 1 ERC1155 token at 1 eth.
2. They both sign the details and everything looks normal.
3. Before sending the transaction Alice changed the value of the protocol property from the MatchOrder structure from ERC1155 to ERC721.
4. Alice calls the buySingleListing with the correct signatures however the protocol parameter is modified.
5. When the execution logic from _executeMatchOrder will be at line #1306 where it will choose what function to call, because the protocol value is ERC721 instead of ERC1155 will call the transferFrom function, if the transferFrom function will not exist in ERC1155 the execution will revert, however if the ERC1155 will have an empty fallback function implemented, the execution will succeed but in reality no nft's will be transferred.

Recommendation:

Add the property protocol from MatchOrder structure in the both seller and buyer signatures.

POC : https://hackmd.io/@QyPimM2nQzSOYG0jHv2aIg/HkOuspaz3

HIGH | RESOLVED

Edge case in business logic lead to stolen nft

In contract PaymentProcessor.sol, in function _executeMatchOrderSale, when the sale is paid in ether (paymentCoin is address(0)), the contract logic is checking that the offer is accepted by the seller through the flag sellerAcceptedOfer, if that flag is true, the execution will be reverted because it means that the transaction has been originated from the seller, and the seller will pay in ether for his own nft. However, that logic can be manipulated because a malicious marketplace or frontend could put the parameter on false even if the transaction has originated from the seller.

Recommendation:
Remove the parameter from the structure and do that check using blockchain state instead of function parameters. That can be achieved by checking the msg.sender and the tx.origin variables are different from the seller of the nft.

```
if (saleDetails.seller != msg.sender && saleDetails.seller != tx.origin) {
    revert PaymentProcessor__CollectionLevelOrItemLevelOffersCanOnlyBeMadeUsingERC20PaymentMethods();
}
```

POC: https://hackmd.io/@QyPimM2nQzSOYG0jHv2aIg/BJzo-C6zh

HIGH | RESOLVED

Using transferFrom can lead to edge cases when the token is an weird implementation of erc20

In contract PaymentProcessor.sol, in functions _payoutCoinCurrency, _computeAndDistributeProceeds, the transfer of tokens is done by simply calling the transferFrom function, however this is a bad implementation decision because the erc20 is an old standard and there are some erc20 tokens that will not revert the execution in case of an failed transfer and they will only return false, as the contract logic is not checking for the transferFrom function output the contract execution will proceed even if the tokens transfer in reality have failed.

Recommendation:
Use the safeTransferFrom function from the SafeERC20 library instead of plain transferFrom.

POC : https://hackmd.io/@QyPimM2nQzSOYG0jHv2aIg/HyllXJCGn

MEDIUM | RESOLVED

No sanity checks when setting a security policy to a collection

In contract PaymentProcessor.sol, in function setCollectionSecurityPolicy, when a security policy id is set there is no check that a security policy corresponding to that id actually exists. This issue can lead to undefined behaviors as a collection owner can set an uninitialized security policy by mistake.

Recommendation:
Add a boolean property in the security policy structure called isCreated, that default will have the false value and turn to true only when the security policy is created, then check in the function setCollectionSecurityPolicy that property is true.

Logic edge case if a chain re-organisation happens

In contract PaymentProcessor.sol, the flow is firstly checking if the enforcePricingConstrains is enable and if it is enable is checking if the price is set between floor price and ceiling price, if it is not enabled then the execution will continue without checking if the price is between the given values, the floor price and ceiling price are set in the "setTokenPricingBoundaries" function and the enforcePricingConstrain is set during the security policy update or create flows, that means that the enabling of eforcePricingConstrain and the setting of the floor price and ceiling price can be set in different transaction, in a case of a chain re-organisation this can be problematic, let's explore the following scenario:

Scenario #1 (normal flow):

1. Bob setup his collection, creating a security policy where he enables the price constrain enforce in tx #1 block #1
2. Bob setup his pricing bound by calling the function "setTokenPricingBoundaries" in tx#2 block #1
3. Alice, Carol & David execute different nft transactions using PaymentProcessor over Bob collection, creating tx #3, #4 & #5 that will also get indexed in block #1.

Scenario #2 (chain re-organisation happens):

1. Bob setup his collection, creating a security policy where he enables the price constrain enforce in tx #1 block #1
2. Alice, Carol & David execute different nft transactions using PaymentProcessor over Bob collection, creating tx #2, #3 & #4 that will also get indexed in block #1.
3. Bob setup his pricing bound by calling the function "setTokenPricingBoundaries" in tx#5 block #1.

Because the the transaction got re-organisat inside the latest block and then between the enabling of enforcing price constrain transactions and the setting of floor price & ceiling price we're putted other transactions that we're executing order on PaymentProcessor, those transactions that got between them have been reverted as they are not matching the ceilingPrice value, which will now be `0` by default.

Recommendation:

When enabling the price constrains also setup the floor price and ceiling price in the same function.

Note #1 : The Limit Break team is acknowledging the possibility of this issue but take no action to remediate. Chain reorganisation attacks are generally extremely rare and findings for them should be evaluated solely when the consequences are devastating (I.e. total loss of funds). Furthermore, Limit Break teams is expecting the best practice around managing security policies and pricing constraints to be to use a multi-sig, which is capable of executing all of the transactions needed to configure pricing constraints in a single transaction context. This means owners of NFT collections seeking to use the features can simply follow security best practices and actively mitigate this minor risk themselves. The impact of a reorganization edge case as described is negligible.

MEDIUM | ACKNOWLEDGED

DOS through sandwich attack on ceilingPrice

In contract PaymentProcessor.sol, function setTokenPricingBounds is used to set floorPrice and ceilingPrice for certain tokenIds, there is only one sanity check at line #571 that will revert if floorPrice is bigger then ceilingPrice, however floorPrice and ceillingPrice can have the same value and execution will continue successfully, allowing the value of ceilingPrice to be set as `0`, this can be leveraged through a sandwich attack if the contract owner/admin whish to deny certain transactions.

Recommendation:
Add a sanity check to ensure that ceilingPrice is different from floorPrice or that ceilingPrice is different from `0`.
Note #1 : Limit Break team intention in this design is that floor and ceiling prices can be the same if the creator wishes (basically like Price Tags for NFTs), and that they can even enforce that NFTs are always free (`0` floor, `0` ceiling). Thus, Limit Break team is inclined to acknowledge the concern but they consider this to be the prerogative of the creator when they are pricing their NFTs.

LOW | RESOLVED

No sanity checks when adding a new coin

In contract PaymentProcessor.sol, in function approveCoin (renamed to whitelistPaymentMethod in latest commit), there are no sanity checks over the coin parameter, if the coin parameter will be set to the zero address or to an address that have no bytecode associated to it all the calls to that address will succeed but in reality nothing will change in the blockchain state, no tokens/value will be transferred.

Recommendation:
1. Add a check to ensure the coin parameter is not address(`0`).
2. Add a check to call the decimals functions from the erc20 (this is a function that all erc20 should implement) and check that the result is different from `0`

LOW | RESOLVED

Missing zero address check for account parameter

The whitelistExchange() function has a missing zero address check for account parameters. This can result in a zero address being accidentally passed in the account parameter.

Recommendation:
It is advised to add a missing zero address check for the account parameter.

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

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security

As a part of our work assisting Limit Break in verifying the correctness of their contract/s code, our team was responsible for writing integration tests using the Hardhat testing framework.

The tests were based on the functionality of the code, as well as a review of the Limit Break contract/s requirements for details about issuance amounts and how the system handles these.

PaymentProcessor

- ✓ Should be able to pause contract (100ms)
- ✓ Should be able to unpause contract (122ms)
- ✓ Should be able to create security policy (74ms)
- ✓ Should be able to update security policy (261ms)
- ✓ Should be able to buy single listing OF ERC1155 (2821ms)
- ✓ Should be able to buy single listing of ERC721 (1437ms)
- ✓ Should be able to buy batch of listings (1001ms)
- ✓ Should be able to buy bundled listing with ERC1155 (3497ms)
- ✓ Should be able to buy bundled listing with ERC721 (757ms)
- ✓ Should be able to sweep collections with ERC1155 (1132ms)
- ✓ Should be able to sweep collection with ERC721 (449ms)
- ✓ Should be able to transfer security policy ownership (136ms)
- ✓ Should be able to renounce security policy ownership (80ms)
- ✓ Should be able to get security policy (101ms)
- ✓ Should be able to get domain sepearator
- ✓ Should be able to whitelist exchange (233ms)
- ✓ Should be able to unwhitelist exchange (189ms)
- ✓ Should be able to approve coin (175ms)
- ✓ Should be able to disapprove coin (178ms)
- ✓ Should be able to set collection security policy (403ms)
- ✓ Should be able to get token security policy id (116ms)
- ✓ Should be able to revoke master nonce (38ms)
- ✓ Should be to revoke single nonce (44ms)
- ✓ Should be able to support interface

24 passing (25s)

The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

FILE	% STMTS	% BRANCH	% FUNCS	% FUNCS	UNCOVERED LINES
PaymentProcessor.sol	98.8	96.19	100	99.18	1891,1900,1909
All Files	**98.8**	**96.19**	**100**	**99.18**	

We are grateful for the opportunity to work with the Limit Break team.

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

Zokyo Security recommends the Limit Break team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo