OPIUM NETWORK STAKING SMART CONTRACT AUDIT

January 15, 2021

opium.network

MixBytes()

CONTENTS

1. INTRODUCTION

1.1 DISCLAIMER

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of Opium Network (name of Client). If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 PROJECT OVERVIEW

The Opium protocol is a universal protocol to create, settle and trade virtually all derivatives and financial instruments in a professional and trustless way. It allows anyone to build custom exchange-traded products on top of the Ethereum blockchain. Once created, they can be traded freely via a network of relayers and will be priced according to supply and demand.

1.3 SECURITY ASSESSMENT METHODOLOGY

At least 2 auditors are involved in the work on the audit who check the provided source code independently of each other in accordance with the methodology described below:

01 "Blind" audit includes:
> Manual code study
> "Reverse" research and study of the architecture of the code based on the source code only

Stage goal:
Building an independent view of the project's architecture
Finding logical flaws

02 Checking the code against the checklist of known vulnerabilities includes:
> Manual code check for vulnerabilities from the company's internal checklist
> The company's checklist is constantly updated based on the analysis of hacks, research and audit of the clients' code

Stage goal:
Eliminate typical vulnerabilities (e.g. reentrancy, gas limit, flashloan attacks, etc.)

03 Checking the logic, architecture of the security model for compliance with the desired model, which includes:
> Detailed study of the project documentation
> Examining contracts tests
> Examining comments in code
> Comparison of the desired model obtained during the study with the reversed view obtained during the blind audit

Stage goal:
Detection of inconsistencies with the desired model

04 Consolidation of the reports from all auditors into one common interim report document
> Cross check: each auditor reviews the reports of the others
> Discussion of the found issues by the auditors
> Formation of a general (merged) report

Stage goal:
Re-check all the problems for relevance and correctness of the threat level
Provide the client with an interim report

05 Bug fixing & re-check.
> Client fixes or comments on every issue
> Upon completion of the bug fixing, the auditors double-check each fix and set the statuses with a link to the fix

Stage goal:
Preparation of the final code version with all the fixes

06 Preparation of the final audit report and delivery to the customer.

Findings discovered during the audit are classified as follows:

FINDINGS SEVERITY BREAKDOWN

Level	Description	Required action
Critical	Bugs leading to assets theft, fund access locking, or any other loss funds to be transferred to any party	Immediate action to fix issue
Major	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.	Implement fix as soon as possible
Warning	Bugs that can break the intended contract logic or expose it to DoS attacks	Take into consideration and implement fix in certain period
Comment	Other issues and recommendations reported to/acknowledged by the team	Take into consideration

Based on the feedback received from the Customer's team regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The project team is aware of this finding. Recommendations for this finding are planned to be resolved in the future. This finding does not affect the overall safety of the project.
No issue	Finding does not affect the overall safety of the project and does not violate the logic of its work.

1.4 EXECUTIVE SUMMARY

The audited scope includes a staking mechanism based on Opium protocol. Staking contract allows users to organize and participate in pools of some specific positions on Opium core protocol. The project includes two main modules: a staking module that implements general staking functionality and a position tokenization module that wraps Opium positions into ERC-20 token.

1.5 PROJECT DASHBOARD

Client	Opium Network
Audit name	Staking Smart Contract Audit
Initial version	a1a3518f6c1af90d4c196d1ee76d30f26ce0f8eb
Final version	e5cf6cd8708b221f395b061767fb5c14ec21bbd1
SLOC	332
Date	2020-12-21 - 2021-01-15
Auditors engaged	2 auditors

FILES LISTING

OpiumERC20Position.sol	OpiumERC20Position.sol
OpiumStakingErrors.sol	OpiumStakingErrors.sol
OpiumStakingDerivatives.sol	OpiumStakingDerivatives.sol
OpiumStaking.sol	OpiumStaking.sol

FINDINGS SUMMARY

Level	Amount
Critical	0
Major	2
Warning	3
Comment	1

CONCLUSION

Smart contracts have been audited and several suspicious places have been spotted. During the audit no critical issues were found, two issues were marked as major because they could lead to some undesired behavior, also several warnings and comments were found and discussed with the client. After working on the reported findings all of them were resolved or acknowledged (if the problem was not critical). So, the contracts are assumed as secure to use according to our security criteria.

2. FINDINGS REPORT

2.1 CRITICAL

Not Found

2.2 MAJOR

MJR-1	Potential lock of `hedge` function execution
File	OpiumStakingDerivatives.sol
Severity	Major
Status	Fixed at 25e09749

DESCRIPTION

At line OpiumStakingDerivatives.sol#L177 the contract uses the `safeApprove` method, but this method has some implicit behavior that can lead to call failure in case if the token owner account has already a non-zero allowance. So basically, some unspent allowance can block the `hedge` function execution.

RECOMMENDATION

We recommend using a trick with allowance zeroing before setting a new one to avoid potential issues. The code can look like:

```
underlying.safeApprove(address(opiumTokenSpender), 0);
underlying.safeApprove(address(opiumTokenSpender), requiredMargin);
```

MJR-2	Potential lock of withdrawal in `OpiumERC20Position`
File	OpiumERC20Position.sol
Severity	Major
Status	**Fixed** at **04cdbc77**

DESCRIPTION

The `OpiumERC20Position` contract has the `withdraw` method that burns a position token and transfers an underlying token to `msg.sender`. According to the specification and code, withdrawal is allowed only after position execution that the condition is checked by the following code at lines OpiumERC20Position.sol#L67-L68

```
uint256 contractPositionsBalance = tokenMinter.balanceOf(address(this), tokenId);
require(contractPositionsBalance == 0, ERROR_CANT_WITHDRAW_BEFORE_EXECUTION);
```

So, as we can see the execution is possible only if `contractPositionsBalance` is zero that happens after position execution, but an attacker can use a front-running vector and send a small amount of token to the contract address in each block or before each withdrawal attempt. This vector can block withdrawals for an undefined time period.

Also the same issue is spotted in the `isStaking` modifier of OpiumStakingDerivatives contract here: OpiumStakingDerivatives.sol#L139-L140. Here this issue can lead to blocking all the functions which used this modifier for an undefined time period as well.

RECOMMENDATION

We recommend to call execute in withdraw function.

CLIENT'S COMMENTARY

> We decided to not call execute on withdrawals, because it led to other code changes.

2.3 WARNING

WRN-1	Add a check that `STAKING_PHASE` is less than `EPOCH`
File	OpiumStakingDerivatives.sol
Severity	Warning
Status	**Fixed** at **3ea1e508**

DESCRIPTION

According to epoch lifecycle logic `STAKING_PHASE` should be less than `EPOCH` but that invariant is never checked before initialization: OpiumStakingDerivatives.sol#L59-L60

RECOMMENDATION

We suggest adding a particular check.

WRN-2	Potentially incorrect staking contract initialization
File	OpiumStaking.sol
Severity	Warning
Status	**Fixed** at **9a52b689**

DESCRIPTION

This issue is about the `OpiumStaking` contract that can possibly be initialized incorrectly with a `_derivative` argument being passed without additional validation in here: OpiumStaking.sol#L34.

Incorrect initialization can lead to the corruption of the state, which will require for the contract to be re-deployed.

RECOMMENDATION

It is recommended to introduce an additional check on the input `_derivative` parameter validity using additional application-logic related checks just as it was done in here: OpiumCdsSyntheticId.sol#L37.

WRN-3	Short position execution validation check
File	OpiumStakingDerivatives.sol
Severity	Warning
Status	**Fixed** at **9a52b689**

DESCRIPTION

This issue is about an absent check whether epoch timings and `shortTokenId` are correct for executing short positions within the `OpiumStakingDerivatives` contract.

Using `execute()` defined at OpiumStakingDerivatives.sol#L118 right after the contract initialization with uninitialized `shortTokenId` (which would also mean no `initializeEpoch` was called) may lead to the incorrect contract behavior from the application logic point of view.

RECOMMENDATION

It is recommended to introduce additional checks on the `shortTokenId` variable to make sure the epoch was initialized and there will be no attempt to execute short positions with incorrect data usage.

2.4 COMMENTS

CMT-1	Extra `/` in imported file path
File	OpiumStakingDerivatives.sol
Severity	Comment
Status	**Fixed** at **be06b198**

DESCRIPTION

At line OpiumStakingDerivatives.sol#L16 there is an extra `/` in the imported file path:

```
import "../PositionsTokenization//OpiumERC20Position.sol";
```

RECOMMENDATION

We suggest removing the extra `/` character.

3. ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build open-source solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

BLOCKCHAINS

 Ethereum

 Cosmos

 EOS

 Substrate

TECH STACK

 Python

 Solidity

 Rust

 C++

CONTACTS

 https://github.com/mixbytes/audits_public

 https://mixbytes.io/

 hello@mixbytes.io

 https://t.me/MixBytes

 https://twitter.com/mixbytes