Published in SmartDec Cybersecurity Blog

Ivan Ivanitskiy Follow

Mar 24, 2020 · 10 min read · Listen

Opium Smart Contracts Security Analysis

SmartDec

In this report, we consider the security of the Opium project. Our task is to find and describe security issues in the smart contracts of the platform.

Disclaimer

The audit does not give any warranties on the security of the code. One audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Besides, security audit is not an investment advice.

Summary

and a number of low severity issues were found. We discussed them with the developers, so in the latest version of the code they fixed most of the issues found in the audit and provided their comments on the rest of the issues.

General recommendations

The contracts code is of good code quality. Thus, we do not have any additional recommendations.

Procedure

In our audit, we consider the following crucial features of the smart contract code:

1. Whether the code is secure.
2. Whether the code corresponds to the documentation (including whitepaper).
3. Whether the code meets best practices in efficient use of gas, code readability, etc.

We perform our audit according to the following procedure.

Automated analysis

- we scan project's smart contracts with our own Solidity static code analyzer SmartCheck
- we scan project's smart contracts with several publicly available automated Solidity analysis tools such as Ethlint and Solhint
- we manually verify (reject or confirm) all the issues found by tools

Manual audit

- we manually analyze smart contracts for security vulnerabilities
- we check smart contracts logic and compare it with the one described in the documentation
- we run tests

Report


We have scanned Ethex smart contracts for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered (the full list includes them but is not limited to them):

- Reentrancy
- Front running
- DoS with (unexpected) revert
- DoS with block gas limit
- Gas limit and loops
- Locked money
- Integer overflow/underflow
- Unchecked external call
- ERC20 Standard violation
- Authentication with tx.origin
- Unsafe use of timestamp
- Using blockhash for randomness
- Balance equality
- Unsafe transfer of ether
- Fallback abuse
- Using inline assembly
- Short address attack
- Private modifier
- Compiler version not fixed


- Implicit visibility level
- Use delete for arrays
- Byte array
- Incorrect use of assert/require
- Using deprecated constructions

Project overview

Project description

In our analysis we consider Opium specification (”docs/index.md” in the project repository) and smart contracts’ code (private repository, version on commit 60ff6f80996b83f6ad19c35b74480fef34f7dc03). Also, the main dependency — erc721o (version on commit 60723b713e3f3f9d98d71b800673e0b4696fe108) was analysed.

The latest version of the code

After the initial audit, some fixes were applied and the code was updated to the latest version (commit 8c7d076fa0ee958a82134c32412d972452746dff).

Project architecture

For the audit, we were provided with the truffle project. The project is an npm package and includes tests.

- The project successfully compiles with truffle compile command (with some warnings, see Compilation output in Appendix)
- The project successfully passes all the tests, however, code coverage was not generated

The total LOC of audited Solidity sources is 1569.

Manual analysis

The contracts were completely manually analyzed, their logic was checked and

Critical issues

Critical issues seriously endanger smart contracts security. We highly recommend fixing them.

The audit showed no critical issues.

Medium severity issues

Medium issues can influence smart contracts operation in current implementation. We highly recommend addressing them.

Overpowered role

The governor address is responsible for whitelisting addresses in TokenSpender contract. That means that governor address is responsible for Opium Token transfers.

So, the system depends heavily on governor address in the current implementation. In this case, there are scenarios that may lead to undesirable consequences for investors, e.g. if the governor's private keys become compromised. Thus, we recommend designing contracts in a trustless manner.

> *Comment from the developers: "TokenSpender contract is used to hold user's allowances on ERC20 contracts and allows only whitelisted contracts to spend tokens. For purpose of users security we introduced "time locks", which notify users when content of whitelist is about to change and doesn't allow to change it before specific moment in future, so users could verify newly proposed whitelist and decide whether they want to keep their allowance or not."*

Bug

There is a bug in SwaprateMatchBase.sol file, line 62. address(0) is used instead of _token address to transfer tokens:

```
ERC20(address(0)).transfer(msg.sender, balance);
```

We recommend fixing this bug (i.e. calling _token address).


There is no deployment script in the project. However, the contracts deployment does not seem trivial. Bugs and vulnerabilities often appear in deployment scripts and severely endanger system's security.

We highly recommend developing and testing deployment scripts very carefully.

> *The issue has been fixed and is not present in the latest version of the code.*

Moreover, there are a lot of setter-functions in Registry.sol (lines 44–89).

We recommend moving the complexity of the project deployment and initialization from smart contracts to deployment scripts in order to reduce gas costs, and removing set***() functions from ABI, as they will be used only once. Bounded smart contracts can be deployed as described in this article.

> *Comment from the developers: "We didn't implement bounding contracts like described in your article, but eliminated all setter to one init() function in Registry contract."*

Low severity issues

Low severity issues can influence smart contracts operation in future versions of code. We recommend taking them into account.

Compiler version

Solidity source files indicate the versions of the compiler they can be compiled with. Example:

```
pragma solidity ^0.5.4; // bad: compiles w 0.5.4 and above
pragma solidity 0.5.4; //good: compiles w 0.5.4 only
```

We recommend following the latter example, as future compiler versions may handle certain language constructions in a way the developers have not foreseen. Besides, we recommend using the latest compiler version — 0.5.16 at the moment.

> *The issue has been fixed and is not present in the latest version of the code.*


- WhitelistedWithGovernance.sol, line 53 whitelist variable is used as the Committed event argument. We recommend using _whitelist variable instead.

> *The issue has been fixed and is not present in the latest version of the code.*

- WhitelistedWithGovernance.sol, line 59 proposedWhitelist variable is used as the Proposed event argument. We recommend using _whitelist variable instead.

> *The issue has been fixed and is not present in the latest version of the code.*

- Core.sol, line 521 registry.getOpiumAddress() is called multiple times. We recommend saving it to the local memory after the first call. The same issue arises at MatchLogic.sol, lines 159, 166.

> *The issues have been fixed and are not present in the latest version of the code.*

- MatchLogic.sol, line 151 registry.getTokenSpender() is called multiple times. We recommend saving it to the local memory after the first call.

> *The issue has been fixed and is not present in the latest version of the code.*

- Whitelisted.sol, line 14 whitelist.length is read in a loop. We recommend saving it to the local memory.

> *The issue has been fixed and is not present in the latest version of the code.*

- Whitelisted.sol, line 9 onlyWhitelisted() modifier consumes a lot of gas due to lots of storage read operations. Keep in mind, that the price of reading storage variables can be changed in the future. For example, it was increased 4 times in the Istanbul hardfork.

Code logic

- WhitelistedWithGovernance.sol, line 50 initialized variable is used to check whether whitelist has been initialized or not. We recommend using whitelist.length instead.

> *The issue has been fixed and is not present in the latest version of the code.*

 

> *Comment from the developers: "In places, where this function is called internally we consider _derivativeHash as trusted and use it to retrieve Ticker state without recalculating the hash and optimise gas usage."*

- MatchSwap.sol, lines 137, 143 When calculating order's filled percentage, 1 is added to every calculated value. We recommend removing these additions.

> *The issue has been fixed and is not present in the latest version of the code.*

Missing input validation

There are input validation issues in several places:

- WhitelistedWithGovernance.sol, line 81 In case _governor argument is zero address, whitelisting functionality will be lost.
- Registry.sol, line 86 In case _opiumAddress argument is zero address, fees will be locked.

We recommend adding check that _governor and _opiumAddress arguments are non-zero.

> *The issues have been fixed and are not present in the latest version of the code.*

Redundant code

The following lines are redundant:

- WhitelistedWithGovernanceAndChangeableTimelock.sol, line 13

```
uint256 timelockProposalTime = 0;
```

- WhitelistedWithGovernanceAndChangeableTimelock.sol, line 15

```
uint256 proposedTimelock = 0;
```


- WhitelistedWithGovernance.sol, line 25

```
uint256 public proposalTime = 0;
```

At these lines, the initial values are reassigned.

We highly recommend removing redundant code in order to improve code readability and transparency and decrease cost of deployment.

> *The issues have been fixed and are not present in the latest version of the code.*

Visibility level

`registry` variable of usingRegistry contract has private visibility level without getter function. This may affect user experience in a negative way as users cannot easily check the registry address.

> *The issue has been fixed and is not present in the latest version of the code.*

Implicit visibility level

The following variables have an implicit visibility level:

- ExecutableByThirdParty.sol, line 6

```
mapping (address => bool) thirdpartyExecutionAllowance;
```

- LibCommission.sol, line 6

```
uint256 constant COMMISSION_BASE = 10000;
```

- LibCommission.sol, line 9


- LibCommission.sol, line 12

```
uint256 constant OPIUM_COMMISSION_PART = 1;
```

- WhitelistedWithGovernanceAndChangableTimelock.sol, line 13

```
uint256 timelockProposalTime = 0;
```

- WhitelistedWithGovernanceAndChangableTimelock.sol, line 15

```
uint256 proposedTimelock = 0;
```

- SwaprateMatch.sol, line 17

```
mapping (bytes32 => uint256) filled;
```

- SwaprateMatchBase.sol, line 31

```
mapping (bytes32 => bool) canceled;
```

- SwaprateMatchBase.sol, line 36

```
mapping (bytes32 => bool) verified;
```

- SwaprateMatchBase.sol, line 44


We recommend specifying visibility levels (public, private, or internal) explicitly and correctly in order to improve code readability.

> *The issues have been fixed and are not present in the latest version of the code.*

Misleading name

validateCanceled() function of SwaprateMatchBase contract checks whether the order is not canceled.

We recommend renaming it to validateNotCanceled().

> *The issue has been fixed and is not present in the latest version of the code.*

Transfer of an unknown ERC20 token

Return value of transfer() function is ignored in the following places:

- MatchLogic.sol, line 70
- Core.sol, line 64
- SwaprateMatchBase.sol, line 62

According to ERC20 token standard:

```
The function SHOULD throw if the message caller's account balance
does not have enough tokens to spend.
```

Since the code of this token contract is unknown, transfer() function call may not revert in the case of unsuccessful token transfer. We recommend using SafeERC20 contract from OpenZeppelin library.

> *The issues have been fixed and are not present in the latest version of the code.*

Code duplications

There are code duplications in the contracts, for example at MatchSwap.sol, lines 133–137 and 139–143.


> *Comment from the developers: "We consider refactoring not feasible, thus we agreed to keep it as is."*

Constant state variable

commission variable at HasCommission.sol, line 8 is never changed.

We recommend adding constant keyword and naming it in UPPER_CASE.

> *The issue has been fixed and is not present in the latest version of the code.*

Prefer external to public visibility level

In the code, there are functions with the public visibility level that are not called internally.

We recommend changing visibility level of such functions to external in order to improve code readability. Moreover, in many cases functions with external visibility modifier require less gas comparing to functions with public visibility modifier.

> *Comment from the developers: "We consider refactoring not feasible, thus we agreed to keep it as is in most of the places where issue occurs."*

Notes

Naming convention

- WhitelistedWithGovernance.sol, line 16 TIME_LOCK_INTERVAL state variable is not constant, but is named in UPPER_CASE_WITH_UNDERSCORES.
  We recommend naming it in mixedCase.
- usingRegistryErrors.sol, line 3 usingRegistryErrors contract is named in mixedCase.
  We recommend naming it in CapWords.
- usingRegistry.sol, line 8 usingRegistry contract is named in mixedCase.
  We recommend naming it in CapWords.

> *The issues have been fixed and are not present in the latest version of the code.*


the execution of the corresponding functions will fail due to an out-of-gas exception.

> *Comment from the developers: "By conducting tests we consider refactoring not feasible, thus we agreed to keep it as is."*

Avoid using experimental features in production code

ABIEncoderV2 is used in the code. We do not recommend using experimental features in the code since they might contain bugs that will only be fixed in future versions of the compiler.

> *Comment from the developers: "We are widely using passing structures as function arguments, thus ABIEncoderV2 is required for us and we can't avoid it."*

Gas limit and loops

The following loops traverse through arrays of variable length:

- Core.sol, line 300

```
for (uint256 i; i < tokenIds.length; i++)
```

- Core.sol, line 344


```
for (uint256 i; i < tokenIds.length; i++)
```

The traversed arrays are passed as parameters of the functions. Therefore, if there are too many items in these arrays, the execution of the corresponding functions will fail due to an out-of-gas exception.

In these cases, we recommend separating the calls into several transactions.

> *This audit was performed by SmartDec, a security team specialized in static code analysis, decompilation and secure development.*

SmartDec

More from SmartDec Cybersecurity Blog

Follow

Security tutorials, tools, and ideas

Read more from SmartDec Cybersecurity Blog

Recommended from Medium

Mayfly

Emrata's NFT

Karbo Cryptocurrency

Prevent transaction cancellation in 51%-attack

TimeBit

Fisco Executive Says That It Is The China Dream Which Is Hindering Digital Yuan

EYWA Protocol

What EYWA gives to Liquidity Providers?

Crypton Studio in Coinmonks

Crypton Studio is pleased to announce a partnership with Polylastic for Advanced Index Development

Saurav Dhungana

ERC20 price prediction: you should Buy or sell Erc20?

Heinrich Bredenkamp

The Winklevoss' bitcoin exchange is pushing into banking with a new savings product