SOLIDIFIED

Audit Report for Dopex - June 14, 2021

Summary

Audit Report prepared by Solidified covering the Dopex options protocol smart contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code in several rounds. The debrief took place on 24 May 2021.

Audited Files

The source code has been supplied in an audit-specific branch of the following source code repository:

https://github.com/dopex-io/core-contracts/tree/audit

UPDATE: Fixes were received on 31 May 2021

Commit number: `7aa0f21e32e8acc9a0c4f219acc45eceef23ee75`

The scope of the audit was limited to the following files:

```
contracts
├── MockContract.sol
├── dopex
│   ├── Dopex.sol
│   ├── asset-swapper
│   │   ├── AssetSwapper.sol
│   │   └── IAssetSwapper.sol
│   ├── delegator
│   │   └── Delegator.sol
│   ├── governance
│   │   └── GovernanceStaking.sol
│   ├── interfaces
│   │   ├── IUniswapV2Router01.sol
│   │   └── IUniswapV2Router02.sol
│   ├── libraries
│   │   └── OptionPoolHelper.sol
│   ├── margin
│   │   └── Margin.sol
│   ├── options
│   │   ├── OptionsContract.sol
│   │   ├── OptionsFactory.sol
│   │   └── interfaces
│   │       ├── IOptionsContract.sol
│   │       └── IOptionsFactory.sol
│   ├── oracle
│   │   ├── DopexOracle.sol
```

```
│   │   ├── IVJobRequest.sol
│   │   ├── PriceJobRequest.sol
│   │   ├── interfaces
│   │   │   ├── IDopexOracle.sol
│   │   │   └── IJobRequest.sol
│   │   └── uniswap-oracle
│   │       ├── BlockVerifier.sol
│   │       ├── IUniswapV2Pair.sol
│   │       ├── MerklePatriciaVerifier.sol
│   │       ├── Rlp.sol
│   │       ├── UQ112x112.sol
│   │       └── UniswapOracle.sol
│   ├── pools
│   │   ├── OptionPool
│   │   │   ├── OPCore.sol
│   │   │   ├── OPStats.sol
│   │   │   ├── OPWithdraw.sol
│   │   │   └── OptionPool.sol
│   │   ├── OptionPoolBroker
│   │   │   ├── OptionPoolBroker.sol
│   │   │   ├── OptionPricing.sol
│   │   │   └── libraries
│   │   │       ├── ArrayHelper.sol
│   │   │       ├── BlackScholes.sol
│   │   │       └── OptionPoolBrokerLibrary.sol
│   │   ├── OptionPoolFactory.sol
│   │   ├── OptionPoolRebates.sol
│   │   ├── VolPool.sol
│   │   └── interfaces
│   │       └── IVolPool.sol
│   ├── rewards
│   │   └── DopexRewards.sol
│   └── token
│       ├── DpxToken.sol
│       ├── ERC20Factory.sol
│       ├── RdpxToken.sol
│       └── UsdtToken.sol
├── erc20
│   ├── ERC20.sol
│   └── IERC20.sol
└── libraries
    ├── ABDKMathQuad.sol
    ├── BokkyPooBahsDateTimeLibrary.sol
    └── SafeMath.sol
```

Intended Behavior

The smart contracts implement a permissionless options protocol. Option pools are created which allow users to deposit base and quote assets. The smart contracts include a Uniswap integration, and a governance token and protocol.

Code Complexity and Test Coverage

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

Note, that high complexity or lower test coverage does equate to a higher risk. Certain bugs are more easily detected in unit testing than a security audit and vice versa. It is, therefore, more likely that undetected issues remain if the test coverage is low or non-existent.

Criteria	Status	Comment
Code complexity	High	-
Code readability and clarity	Low	-
Level of Documentation	Low	-
Test Coverage	Medium	-

SOLIDIFIED

Audit Report for Dopex - June 14, 2021

Issues Found

Solidified found that the Dopex contracts contain 2 critical issues, 3 major issues, 5 minor issues, in addition to 2 informational notes.

Two general warnings have been added.

We recommend all issues are amended, while the notes are up to the team's discretion, as they refer to best practices.

Issue #	Description	Severity	Status
1	Complex Architecture with several parts that appear unfinished	Warning	-
2	Mathematical Implementation of Black Scholes/Pricing could not be Validated	Warning	-
3	OptionPool.sol: Inability for Pool to be marked ready until all previous options contracts expire leads to griefing issue	Critical	Acknowledged
4	OPCore: getOrAddOptionsContract() lacks access control, leading to a number of issues	Critical	Resolved
5	Missing checks for ERC-20 return values throughout the codebase	Major	Resolved
6	OptionPricing.sol: Time threshold is always zero and described inconsistently	Major	Acknowledged
7	Protocol is unsafe with ERC-777 tokens and susceptible to malicious ERC-20 implementations	Major	Resolved
8	DpxToken.sol and OPStats.sol: Potentially unbounded loop iteration	Major	Pending
9	IVJobRequest.sol and PriceJobRequest.sol: Chainlink oracles shadow superclass oracle address	Minor	Acknowledged

10	Order of arithmetic operation may decrease precision	Minor	Pending
11	assetSwapper.sol: Payable function may lead to stuck ETH, since it does not process ETH	Minor	Acknowledged
12	Dopex.sol: Governance operations lack zero checks	Minor	Resolved
13	OptionPricing.sol: Penalties tracker is never used	Minor	Acknowledged
14	Duplicate Libraries	Note	-
15	OPWithdraw.sol: Contract inherits unnecessary contracts	Note	-

Warnings

1. Complex Architecture with several parts that appear unfinished

The code organization is very complex, leading to a large number of contracts with a complex inheritance tree (which is a cyclic graph in places). In several places, this seems to be the result of architectural changes. There are also a number of unused constructs (see issues below), contradictory or unrelated comments, and hard-coded values. Whilst the protocol seems well thought out, the implementation seems unfished and not ready for deployment. One particular component that seems unfinished is the governance module. Whilst there are many governance-only functionalities, it is unclear who can call these since the governance contract is unimplemented.

Recommendation
Consider simplifying the overall architecture and targeting a staged release.

Update
Team reply: "*The inheritance issue (which causes the cyclic graphs) you mentioned in this issue clearly would be from the fragmented OptionPool contract. We intended this fragmentation to increase readability but are seeing an inverse effect hence we will resolve into a single large contract. As for the unused constructs and unfinished implementations (as mentioned for Governance), We understand that this would specifically be for the DopexOracle, OptionPricing and Governance contracts. The DopexOracle contract as of now remains a placeholder as we are currently porting our code for Optimisim L2 wherein several components in the existing DopexOracle contract will not function and hence will be replaced. The OptionPricing contract is not finalised yet, we are working with our Partners (Several Different Trading Institutions) to come up with a final form for this contract but it remains incomplete at the moment and is most definitely going to receive a complete overhaul from its current state due to the changes we have made to our pricing model. As for the governance contract, we will be implementing the TimeLock contract from sushiswap - https://github.com/sushiswap/sushiswap/blob/master/contracts/governance/Timelock.sol.*"

2. Mathematical Implementation of Black Scholes/Pricing could not be Validated

The mathematical model used by the options protocol has not been specified outside the codebase. It has therefore not been possible for the auditor team to verify that the implementation corresponds to the specification. The overall arithmetic implementation uses a number of hard-coded constants and has a large number of deeply nested equations. A substantial risk, therefore, remains, in terms of the validity of the implementation. This applies particularly to the pricing model implemented in `OptionPricing.sol`.

Recommendation
Consider providing a clear specification for the mathematical model, using a cleaner nesting model with well-named constants. It is also often a good idea to generate Solidity math implementations from a higher-level programming language, for example, a Python script, particularly when the safeMath library is used.

Update
Team reply: "*The black scholes model is sufficiently complicated and hence the code to implement it is too. The hard coded values are used for computations in the formula in Solidity (and other programming languages), We understand it is hard to make out and are in the process of refactoring the code with easier to understand variable names and comments at each point of the calculation to reflect that indeed this is the correct formula for Black Scholes. We will also add relevant references to the contract for better understanding.*"

Critical Issues

3. `OptionPool.sol`: Inability for Pool to be marked ready until all previous options contracts expire leads to griefing issue

The fact that an `OptionPool` contract cannot be marked as ready until all contracts in the previous epoch are manually set to expire can be griefed by endlessly entering into low-value options each epoch that cost more in gas to execute than the amount redeemed from their vaults. Even with the `minimumOptionPrice` guard, this can create an arbitrary gas burden to finalize an epoch.

Recommendation
The functionality should be modified such that the entire pool's funds are not locked from withdrawal if not all previous contracts have been manually marked as expired.

Update
Team reply: "*We have already started work towards figuring out a better solution for the bootstrapping process. An initial solution to this problem we have thought about is to change the way collateral is locked into an options contract. Since the OptionPool is the only contract that is able to interact with the OptionsContract (to create state changes) we are thinking of tracking collateral locked for each OptionsContract directly inside of the OptionPool rather than transferring it to the OptionsContract. This way redeeming of the vault balance would not be needed and hence it would eliminate the requirement of expiring option contracts completely.*"

4. OPCore: getOrAddOptionsContract() lacks access control, leading to a number of issues

---

The function `getOrAddOptionsContract()` is completely unprotected and can be called by anyone. This could be exploited in a grievance attack, by someone adding a large number of option contracts without providing collateral, and rendering the epoch unrealizable due to making `redeemVaultBalance()` revert. This is particularly critical since option contract ids can collide meaning several option contracts with the same base- and quote assets cannot exit for the same timeframe.

Recommendation
Limit the function to be only callable from the broker contract.

Major Issues

5. Missing checks for ERC-20 return values throughout the codebase

---

Throughout the codebase, the ERC-20 return values are ignored when making token transfers. Instead, the code assumes that ERC-20 token implementations revert on error. However, this is

not always the case and there are a large number of tokens that return false instead of reverting.
This is an issue when interacting with external tokens.

Recommendation
It is recommended to check the return value of token transfers or use OpenZeppelin's `SafeERC20` extension.

6. OptionPricing.sol: Time threshold is always zero and described inconsistently

The variable `timeThreshold` is never initialized and always remains zero, resulting the following statement used when a quote is proposed never applying:

```
 if (timeThreshold >
block.timestamp.sub(proposedQuotes[msg.sender][_optionPool].lastUpdated))
```

In addition, the comment in the variable declaration does not match the comment in the use of the variable, casting doubts on its purpose.

Recommendation
initialize the variable and clarify its usage.

Update
Team reply: "*As mentioned in issue 1, the OptionPricing contract is a work in progress and this variable is not finalised and may not exist in the final version of the contract.*"

7. Protocol is unsafe with ERC-777 tokens and susceptible to malicious ERC-20 implementations

The protocol makes several calls to external tokens that are susceptible to reentrancy vulnerabilities due to state changes after the external call. This makes the protocol unsafe in conjunction with ERC-777 tokens due to the hooks that may be used to inject code and more vulnerable to malicious tokens. Note, that not all side effects of malicious token implementations are avoidable but preventing reentrancy protection would significantly reduce the risk.

Reentrancy is an issue in the following functions:

`OptionsContract.addERC20Collateral()`
`OPWithdraw.emergencyWithdrawFromPool()`
`OptionsContract.purchase()`

Recommendation
Avoid state changes after external calls to untrusted code or implement a reentrancy guard.

8. DpxToken.sol and OPStats.sol: Potentially unbounded loop iteration

---

`_moveDelegates()` in `DpxToken` and various functions in `OPStats` iterate over loops that can potentially become unbounded depending on user behavior, causing the calling code to be unable to execute.

Recommendation
Implement clear bounds for functions that loop over a range to ensure there is no scenario where the function can fail to execute.

Minor Issues

9. IVJobRequest.sol and PriceJobRequest.sol: Chainlink oracles shadow superclass oracle address

---

The two contracts inherit from `ChainlinkClient` but shadow the `oracle` state variable. Whilst this seems to have no adverse effect on the way the oracle is invoked by specifying the oracle address explicitly, it means that the underlying oracle address is never initialized, which could lead to some oracle methods not functioning as expected.

Recommendation
Consider setting the superclass oracle address and changing the variable name to avoid shadowing for better maintainability.

Update
Team reply: "*The DopexOracle contract will undergo an overhaul once our Optimisim L2 porting is complete. It will not include ChainLink as ChainLink is not currently deployed on Optimisim or for that matter has any plans to.*"

10. Order of arithmetic operation may decrease precision

Throughout the codebase, multiplications are performed on the result of division. In integer arithmetic, this decreases the precision slightly. In some of these cases, the operations are performed over several lines of code and the small decrease in efficiency might be acceptable for improved readability. However, in other cases, the operation is performed in a single statement and reversal would not affect readability.

Recommendation
Consider reversing the order of operations to increase precision where appropriate.

11. assetSwapper.sol: Payable function may lead to stuck ETH, since it does not process ETH

The function `swapAsset()` is declared `payable`. However the function does not allow swapping ETH, nor does the contract provide a means of withdrawing any ETH sent to it, meaning that any ETH sent to this contract will be stuck.
The comment
`// Trade 2: Execute swap of the ERC20 token back into ETH on Sushiswap to complete the arb`
indicates that this might be due to code reuse from a different source.

Recommendation
Consider removing the `payable` keyword.

12. Dopex.sol: Governance operations lack zero checks

The functions setting various essential parameters by the governance contract do check for `address(0)`. This is not an important issue, since most operations can be undone, however in

the case of `setGovernanceContract()`, an `address(0)` parameter would irrevocably disable governance.

Recommendation
Consider adding zero checks, at least in `setGovernanceContract()`.

13. OptionPricing.sol: Penalties tracker is never used

The mapping of `penalties` is incremented if a user proposes after the threshold, however, the penalty "score" is never used. The mapping does not have an access modifier, so by default is private.

Recommendation
Consider either removing the penalty tracker or implement the restrictive consequences on-chain.

Informational Notes

14. Duplicate Libraries

The codebase includes duplicate libraries. Several versions of SafeMath and ERC20 base implementations are included. This increases the complexity of the codebase and complicates maintainability.

Recommendation
Consider cleaning up the codebase to use a single version of libraries for improved maintainability.

15. OPWithdraw.sol: Contract inherits unnecessary contracts

The `OPWithdraw` contract inherits from both the `OPCore` and `OPStats`. This is unnecessary as `OPStats` inherits `OPCore`.

Recommendation

Consider removing unnecessary inheritance.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Dopex or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Solidified Technologies Inc.