DERIVE

Derive (Formerly Lyra) - Round 2

Smart Contract Security Review

Version: 2.0

September, 2024

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Derive smart contracts. The review focused solely on the security aspects of the Solidity implementation of the contract, though general recommendations and informational comments are also provided. A previous security assessment targeting the Derive smart contracts was performed by Sigma Prime in Q3 2023.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the smart contract. Sigma Prime makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the Derive smart contracts contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the Derive smart contracts.

Overview

Derive is a decentralized financial technology protocol designed for options trading, allowing users to optimize trading strategies, hedge risks, and earn yield. The protocol features three main components: managers, liquidations, and cash.

1. **Managers –** There are two types of managers in the Derive protocol: The Portfolio Margin Risk Manager (PMRM) and the Standard Manager.

   PMRM - This manager is designed for professional traders and market makers. Portfolio margin works by aggregating the risk across the entire account, letting the risks of various positions cancel each other out, thereby reducing the overall margin requirements.

   Standard Manager - This manager is designed for the majority of users, catering mostly to non-professional traders or market makers.

2. **Liquidations –** These occur to mitigate the risk of an account becoming insolvent when it falls beneath its margin requirements. If a user falls beneath their margin requirement, their account becomes subject to liquidation. Any user can initiate this process, which may result in the partial or complete liquidation of the account at risk. This is done through an auction process.

3. **Cash –** Cash refers to the designated cash asset, in V2.0, this is USDC. This system facilitates an inherent lending market on the cash asset.

Security Assessment Summary

This review was conducted on the files hosted on the derivexyz GitHub organisation and were assessed at the following commits:

- `v2-core`: 63fff03
- `lyra-utils`: f64d5e3
- `v2-matching`: daf7cdc

Note: the OpenZeppelin libraries and dependencies were excluded from the scope of this assessment.

During testing, a number of changes were made to the protocol, either as a result of identified findings or necessary protocol improvements. As such, the following changes, strictly limited to the contracts in scope, were also reviewed:

- b15187c
- c81cdc3
- 234f5a1
- 61a8391
- 541903c
- 86d48ce
- f21f3e1
- 7152616
- 99cb6b4
- 1d21542
- c9d9480
- 444b035
- 6510200
- 10bb747
- 3b4a0c2
- 92f73bb
- b7d1327
- 254e39b
- fa2fe13
- 41365e3
- aa104cd

The manual code review section of the report is focused on identifying issues/vulnerabilities associated with the business logic implementation of the contracts. This includes their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Ethereum Virtual Machine (for example, verifying correct storage/memory layout). Additionally, the manual review process focused on all known Solidity anti-patterns and attack vectors. These include, but are not limited to, the following vectors: re-entrancy, front-running, integer overflow/underflow and correct visibility specifiers. For a more thorough, but non-exhaustive list of examined vectors, see [1, 2].

To support this review, the testing team used the following automated testing tools:

- Mythril: `https://github.com/ConsenSys/mythril`
- Slither: `https://github.com/trailofbits/slither`
- Surya: `https://github.com/ConsenSys/surya`

Output for these automated tools is available upon request.

Findings Summary

The testing team identified a total of 10 issues during this assessment. Categorised by their severity:

- Critical: 1 issue.

- High: 2 issues.
- Medium: 1 issue.
- Low: 5 issues.
- Informational: 1 issue.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Derive smart contracts. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the contracts, including gas optimisations, are also described in this section and are labelled as “informational”.

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
DRV2-01	Bids Can Be Blocked By Depositing Tokens To Liquidator	Critical	Closed
DRV2-02	Withdrawals Can Be Forcefully Initialised By Anyone	High	Resolved
DRV2-03	Precision Loss in `CashAsset.withdraw()`	High	Open
DRV2-04	`smFeePercentage` Not Cached Properly When Socialised Multiple Times	Medium	Resolved
DRV2-05	Withdrawals Temporarily Blocked By Incorrect Accounting	Low	Closed
DRV2-06	Insolvent Auction Bids Lack Complete `priceLimit` Control	Low	Closed
DRV2-07	Auction Bids Will Work On Both Types Of Auction	Low	Closed
DRV2-08	Insolvent Auction Bids Can Lose Bidding Incentive	Low	Closed
DRV2-09	EIP-712 Compliance Issues	Low	Resolved
DRV2-10	Miscellaneous General Comments	Informational	Open

DRV2-01	Bids Can Be Blocked By Depositing Tokens To Liquidator		
Asset	`DutchAuction.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description

Liquidator bids can be blocked by being front-run with a call to deposit an asset to the liquidator's account, when bidding on auctions `_ensureBidderCashBalance()` is called to ensure a liquidator only has cash in their account. The check is required for gas reasons as then no portfolio risk check has to be performed on the liquidator.

This leads to a problem where any account can front-run a bid by sending a non-cash asset to the bidder/liquidator, which results in the bid reverting. This issue was originally highlighted in `DRV-02` in the previous security assessment report, and fixed for Options assets by requiring approval from the receiving account prior to transfer.

However, no approval is required for depositing/wrapping fresh ERC20 tokens into a recipient account, using `WrappedERC20Asset.deposit()`. As such, it is still possible to send a non-cash asset (wrapped ERC20 in this case) to an arbitrary account, which means this vulnerability is still exploitable.

Recommendations

Extend the approval system to wrapped token deposits.

Resolution

The development team has acknowledged the issue with the following comment:

> *"As wrapper contracts can be created to atomically create new accounts with cash only and then bidding, this will not be changed."*

DRV2-02	Withdrawals Can Be Forcefully Initialised By Anyone		
Asset	`SubAccountsManager.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: Medium	Likelihood: High

Description

The `completeWithdrawAccount()` function can be called without `requestWithdrawAccount()` being called first.

If `requestWithdrawAccount()` is not called first, `withdrawTimestamp[accountId]` will be 0 by default and, as a result, the `WITHDRAW_COOLDOWN` check on line [**86**] will be trivially bypassed.

Considering there are no access controls on `completeWithdrawAccount()`, an attacker could enumerate all account IDs (which would be a trivial task, given they are all sequential) and forcefully withdraw all `SubAccounts` in the system.

Additionally, `requestWithdrawAccount()` does not have a mechanism in place to check if a withdrawal request has already been placed, therefore every consecutive call to `requestWithdrawAccount()` will simply reset and further extend cooling down period.

Note, no funds are at risk as they would simply be transferred back to their original owner's address.

Recommendations

When calling `requestWithdrawAccount()`, perform a check to ensure `requestWithdrawAccount()` has been called first.

Resolution

This issue has been resolved in 254e39b.

DRV2-03	Precision Loss in `CashAsset.withdraw()`		
Asset	`CashAsset.sol`		
Status	**Open**		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

`CashAsset.withdraw()` appears to have some precision loss, which can lead to reverts when using stable assets with native decimals larger than 18 digits.

The `withdraw()` function converts requested `stableAmount` by calling `to18DecimalsRoundUp()` on it, which rounds up the result to closest 18th decimal place in an attempt to clear any dust.

This result is then passed to `_withdrawCashAmount()`, which converts that amount back to 18 decimals via `from18Decimals()` for withdrawal.

As such, it will attempt to withdraw more than available assets, which will result in an unexpected revert.

Recommendations

Do not round up the amounts before withdrawal, instead use the actual amount and, if necessary, implement separate mechanisms to harvest any leftover dust.

DRV2-04	`smFeePercentage` Not Cached Properly When Socialised Multiple Times		
Asset	`CashAsset.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

The `smFeePercentage` is not cached properly when socialised multiple times, resulting in previous and current fees both being the same value.

In `CashAsset.socializeLoss()` if some loss is socialised, it sets the `smFeePercentage` to 100% and stores the previous rate in `previousSmFeePercentage` ready to be restored once the protocol is solvent.

However, if this function is then called a second time, the stored `previousSmFeePercentage` is overwritten with the 100% value, so both current and previous fees will be 100%.

The permissionless recovery function `disableWithdrawFee()` will thus not change the fee from 100% and a non-100% fee will need to be manually reset by a call to `setSmFee()`.

Recommendations

Implement checks to ensure that if the withdrawal fee is already enabled, not to socialise it again.

Resolution

This issue has been resolved in 1d21542.

DRV2-05	Withdrawals Temporarily Blocked By Incorrect Accounting		
Asset	`DutchAuction.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

System withdrawals can become blocked by a stale record of insolvent auction balances. This happens because of `totalInsolventMM`, which records the initial minimum margin of each account with an insolvent liquidation, and is only decreased if an insolvent auction is completely terminated.

Therefore, if a small percentage of an insolvent auction is not bid on due to a liquidator lacking funds or due to malice, then the entire `cachedMM` relating to said auction is not cleared from `totalInsolventMM`. Collecting this dust and terminating the auction would have to be manually performed by the Derive team as it is not worth the gas cost for liquidators. As a result `totalInsolventMM` may keep increasing up to the point where withdrawals are blocked.

Withdrawals can be unblocked by manually setting `smAccount` to 0 to avoid this causing a DoS condition on users. However, this means that the automatic withdrawal blocking feature is lost, and the team would have to manually block and unblock withdrawals on demand.

Recommendations

Decrease `totalInsolventMM` in `bid()` in proportion to the sold amount rather than on total auction completion.

Resolution

The development team has acknowledged the issue with the following comment:

> *"This is only relevant if several large positions are liquidated down to almost nothing. The cost to clear this is minimal (only costs gas) for anyone watching the protocol."*

DRV2-06	Insolvent Auction Bids Lack Complete `priceLimit` Control		
Asset	`DutchAuction.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

Insolvent auction bids have no control over the maximum size of a portfolio they are accepting.

Auction bids may specify a `priceLimit` parameter to reject bids which do not meet their requirements. However, for insolvent bids, because we are dealing with a negative `priceLimit` value, only bids with `cashToBidder - |priceLimit|` will revert on line [**300**].

This is important for risk reasons as when the magnitude of `cashToBidder` is uncapped, the bidder cannot control situations in which the bid size exceeds the security manager's cash balance. These situations would result in cash being printed to the bidder and a withdrawal fee being applied to the system, which may be undesirable to liquidators who wish to immediately exit the system completely after bidding.

Recommendations

Expand the `priceLimit` parameter for insolvent auction bids, allowing liquidators to specify a range of acceptable `cashToBidder` values.

Resolution

The development team has acknowledged the issue with the following comment:

> *"This can be handled in wrapper contracts outside of the protocol, so is acceptable to leave as is."*

DRV2-07	Auction Bids Will Work On Both Types Of Auction		
Asset	`DutchAuction.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

Auction bids intended for a solvent auction can also be applied to an insolvent auction for the same account. This can lead to liquidators bidding on the wrong type of auction and accepting portfolios they did not desire.

Both types of auctions, solvent and insolvent, are processed by calling the `bid()` function. If a solvent auction bid is transmitted, but becomes delayed due to gas costs increasing, then it may be confirmed on-chain later once the auction has been transitioned to an insolvent auction. This can also occur if the portfolio margin risk manager's worst SPAN scenario is updated prior to a bid being included on-chain.

While this function contains two parameters intended to protect the liquidator against bidding conditions, changing neither will protect against this circumstance. As a result, bids intended for a solvent auction can also fulfil an insolvent auction against the same account. A liquidator who intended to buy a portfolio with a positive mark to market may instead accept an insolvent account with a negative mark to market in return for a cash rebate.

Recommendations

Allow a liquidator to specify the type of auction they are intending to bid on with a new bool parameter. If the parameter type does not match the current auction time, the bid should revert.

Resolution

The development team has acknowledged the issue with the following comment:

> *"This can be handled in wrapper contracts outside of the protocol, so is acceptable to leave as is."*

DRV2-08	Insolvent Auction Bids Can Lose Bidding Incentive		
Asset	`DutchAuction.sol, CashAsset.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

During an insolvent auction, it is possible for the exchange rate of cash to USDC to decrease. This decrease may reduce the funds received by the liquidator such that they make a loss.

If the system receives an insolvent auction bid which cannot be covered by the cash in the Security Module, it will print cash. This printed cash can then cause the exchange rate of cash to USDC to decrease such that the USDC withdrawal fee is activated.

This withdrawal fee is dynamic and depends on the ratio of available USDC to the amount of cash in the system. As a result of this, it is possible that the withdrawal fee will be larger than the difference in printed cash and the accepted negative asset value taken on by the liquidator. If this is the case, then it is unlikely any liquidator will bid on these insolvent auctions and the bad debt may sit unresolved in the system.

While possible, this situation would only occur for very large insolvent auction bids, or when the system has little or no funds available in the Security Module, meaning this situation should be less likely to occur as time passes and cash accumulates in the Security Module.

Furthermore, if the system becomes sufficiently insolvent, it is unlikely that any liquidator will bid on insolvent auctions as the impact of further insolvencies compounds the effect of this withdrawal fee as the ratio of USDC to cash degrades further.

Recommendations

The testing team recommends determining the likelihood of these liquidations occurring and developing monitoring tools to detect such liquidations during the early protocol development. It may be beneficial for the Derive team to bid on such auctions if needed.

Alternatively, a successful insolvent liquidation bid could result in a forced USDC withdrawal prior to updating the cash to USDC exchange rate. This would ensure insolvent liquidations would always receive the exchange rate available prior to their bid at the cost of a larger socialised cost to other Derive participants. However, this solution would only solve isolated cases and it is likely that multiple insolvency auctions would occur in the same time period due to the nature of price volatility affecting liquidations.

Resolution

The development team has acknowledged the issue with the following comment:

> *"Given there is a withdrawal pause, there is little incentive for users to not bid on ongoing auctions when they happen at a larger scale. Assets are also marked assuming USDC == 1 within the current risk managers, so the*

values of the portfolios will generally scale with any depeg event. Finally, at the point of the insolvent auction running the full duration, bidders do not even need to bring collateral to bid on auctions, so there will always be someone incentivised to take on those positions."

DRV2-09	EIP-712 Compliance Issues		
Asset	`PermitAllowanceLib.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

`_PERMIT_ALLOWANCE_TYPEHASH` does not comply with the EIP712 standard - *"if the struct type references other struct types (and these in turn reference even more struct types), then the set of referenced struct types is collected, sorted by name and appended to the encoding"* (see eip-712).

There is also a discrepancy between variable names used in `AssetAllowance`, `SubIdAllowance` and their respective typehashes.

Recommendations

Add definition of `AssetAllowance` and `SubIdAllowance` as a part of `_PERMIT_ALLOWANCE_TYPEHASH`.

Rename `delegate` to `asset` in `_ASSET_ALLOWANCE_TYPEHASH` and `_SUBID_ALLOWANCE_TYPEHASH` definitions.

Resolution

This issue has been resolved in 10bb747.

DRV2-10	Miscellaneous General Comments
Asset	`src/*`
Status	Open
Rating	Informational

Description

This section details miscellaneous findings discovered by the testing team that do not have direct security implications.

1. **TradeModule Lacks Zero Index Check**

   Related Asset(s): TradeModule.sol

   In comparison to all other modules, `TradeModule.sol` does not handle the case where `action.subaccountId == 0`. This case is used in the other modules when a new account has to be created.

   Crucially, when verifying an action for a `subaccountId` that does not exist, some verification checks are bypassed (see `ActionVerifier.sol:79`) as such it is trivial to create a valid verified action for subaccount 0. Because `TradeModule.sol` has no extra checks for `subaccountId == 0` it will continue normally. Because no actual subaccount with id 0 exists (it starts counting at 1), it will eventually revert with `ERC721: invalid token ID`.

   It is recommended to catch this edge case earlier by checking if `subaccountId == 0` in `TradeModule.sol` to enforce uniform handling throughout modules.

2. **No Checks to Prevent Negative Value**

   Related Asset(s): BaseManager.sol

   The code on line [**194**] does not check whether `ignoreAmount` is less than `assetBalances[i].balance`. Therefore, the value potentially becomes negative. As the result, the value in `_symmetricManagerAdjustment()` on line [**388**] may become positive. In other words, the amount is transferred from `LiquidatorId` to `accountId`, which is unexpected when liquidating assets.

3. **Dutch Auctions May Deviate From Definition**

   Related Asset(s): DutchAuction.sol

   If the Derive Chain decentralizes their sequencer in the future, it may be possible for transactions to be viewed by some parties prior to their inclusion on-chain. If this occurs, it will weaken the properties of liquidation auctions as Dutch Auctions by definition rely on the fact bidders cannot view each other's bids. If it becomes possible to front-run another bidder, then only the 2nd highest bid price on auctions will be obtained for the protocol.

4. **Centralised Trade Executor**

   Trade executor is a system responsible for placing trades on chain and is one of the key, and highly trusted elements of the protocol. As a potential single point of failure, particular care needs to be taken to ensure its general security posture and access controls, particularly access to and storage of its private keys, are sufficient and appropriately managed.

5. **Additional Bounds Checks**

   Related Asset(s): PMRM.sol

   The `setScenarios()` could benefit from upper limit checks to avoid gas limit exhaustion issues.

   Related Asset(s): UnorderedMemoryArray.sol

   `trimArray()` should ensure that `finalLength < array.length`, otherwise it will extend the array, which will then contain random data from memory. Note, it does not appear to be exploitable based on how it is used in the codebase currently.

`addUniqueToArray()` assumes that a passed in array is large enough and that it has at least 1 slot available at the end of the array to append new elements. It should be surrounded by additional checks, otherwise it will cause unexpected reverts. Note, it does not appear to be exploitable based on how it is used in the codebase currently.

6. **Gas Optimisations**

   Related Asset(s): BaseLyraFeed.sol

   In `_parseAndVerifyFeedData()`, the `BLF_InvalidSigner`, `BLF_DataExpired` and `BLF_InvalidTimestamp` checks can be moved before the signature checks in order to save gas in failed transactions.

   Related Asset(s): Matching.sol

   In `verifyAndMatch()`, the `M_MismatchedModule` check can be performed before calling `_verifyAction()`, which is quite expensive as it verifies signatures, in order to save gas in failed transactions.

7. **Inaccurate Function Names**

   Related Asset(s): BaseLyraFeed.sol

   `addSigner()` function could be renamed to `setSigner()` to better describe its function as it is possible to modify the whitelist by setting (or unsetting) an existing signer.

8. **Stale Comments**

   Related Asset(s): DutchAuction.sol

   Due to changes made in commit f21f3e1 during the review, the comments on line [**317**], line [**319**] and line [**367**] are now incorrectly referring to the original portfolio rather than the current portfolio.

9. **NatSpec Mismatch**

   Related Asset(s): DutchAuction.sol

   Due to changes made in commit f21f3e1 during the review, the NatSpec comments for `_bidOnSolventAuction()` and `_bidOnInsolventAuction()` now reference incorrect function parameters and return values.

10. **TODO Comments**

    Revisit all `TODO` comments throughout the code base, particularly those in test cases encapsulating key functionality or critical modules.

11. **Unfinished Comment**

    Related Asset(s): SubAccountCreator.sol

    Unfinished sentence on line [**46**].

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Appendix A Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

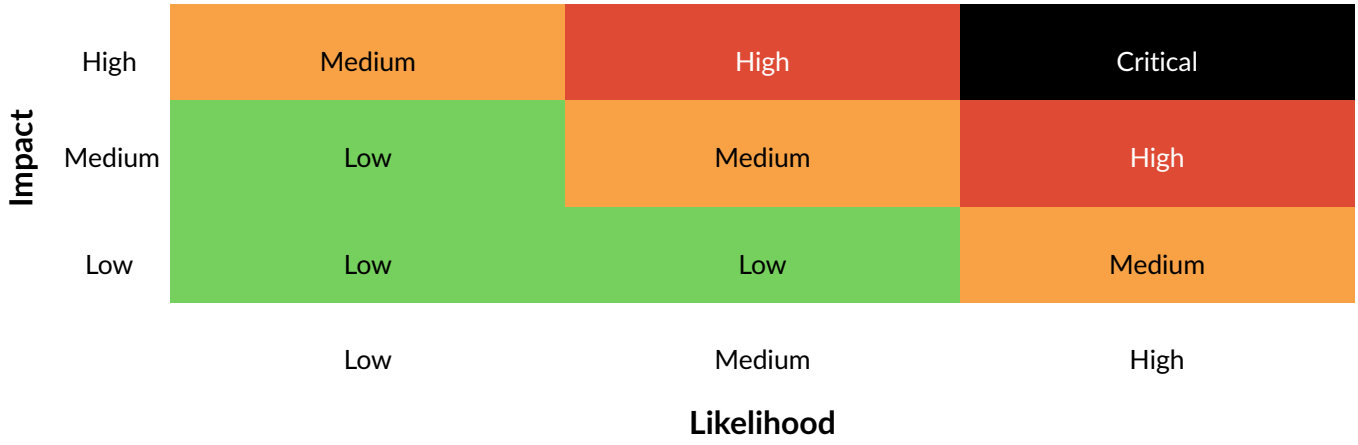

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

References

[1] Sigma Prime. Solidity Security. Blog, 2018, Available: https://blog.sigmaprime.io/solidity-security.html. [Accessed 2018].

[2] NCC Group. DASP - Top 10. Website, 2018, Available: http://www.dasp.co/. [Accessed 2018].