Stella

Smart Contract Security Assessment

July 19, 2023

ABSTRACT

Dedaub was commissioned to perform a security audit of the Stella protocol. Stella is a leveraged strategies protocol with 0% cost to borrow, i.e., borrowers can take leverage on supported DeFi strategies without paying any borrowing cost. Instead, they are required to provide a cut of their profits as yield to the protocol lenders.

The audit covers the whole Stella protocol with the central parts being the lending pool and the strategy contracts, the position managers and the different types of oracles that are being employed by the protocol.

The code and accompanying artifacts (e.g., test suite, documentation) have been developed with high professional standards. No major security issues/threats that could lead to theft or loss of user funds were identified by the audit.

SETTING & CAVEATS

The scope of the audit includes all the core contracts of the Stella protocol and their associated libraries. The test suite was consulted during the audit but was not part of it.

The full list of audited files can be found below:

```
contracts
├── stella-lending
│   ├── common
│   │   ├── FreezeBuckets.sol
│   │   ├── LendingGateway.sol
│   │   ├── RewardVault.sol
│   │   └── RiskFramework.sol
│   ├── lending-pools
│   │   ├── BaseLendingPool.sol
│   │   ├── Erc20LendingPool.sol
│   │   └── NativeLendingPool.sol
│   └── LendingProxy.sol
├── stella-libraries
```

```
│   ├── AccessController.sol
│   ├── BytesLib.sol
│   ├── ProxyAdminImpl.sol
│   ├── TickMath.sol
│   ├── TransparentUpgradeableProxyImpl.sol
│   ├── TransparentUpgradeableProxyReceiveETH.sol
│   ├── UniswapV3Lib.sol
│   ├── UsingAccessController.sol
│   └── UsingAccessControllerUpgradeable.sol
├── stella-oracles
│   ├── AggregatorOracle.sol
│   ├── BandAdapterOracle.sol
│   ├── ChainlinkAdapterOracle.sol
│   ├── UniswapV3Oracle.sol
│   └── UsingBaseOracle.sol
└── stella-strategies
    ├── common
    │   ├── Config.sol
    │   ├── LiquidationVault.sol
    │   ├── ProfitSharingModel.sol
    │   └── StrategyGateway.sol
    ├── factory
    │   └── UniswapV3StrategyFactory.sol
    ├── position-managers
    │   ├── base
    │   │   ├── BasePositionManager.sol
    │   │   └── BasePositionViewer.sol
    │   └── uniswap-v3
    │       ├── UniswapV3PositionManager.sol
    │       └── UniswapV3PositionViewer.sol
    └── strategies
        ├── base
        │   └── BaseStrategy.sol
        ├── SwapHelper.sol
        └── uniswap-v3
            └── UniswapV3Strategy.sol
```

The audit report covers commit hash e70e06d462219bdac76109bec5776ab67f9ff524 of the at the time private repository stella-arbitrum-private-contract.

Two auditors worked on the codebase for 3.5 weeks.

The audit's main target is security threats, i.e., what the community understanding would likely call "hacking", rather than the regular use of the protocol. Functional correctness (i.e. issues in "regular use") is a secondary consideration. Typically it can only be covered if we are provided with unambiguous (i.e. full-detail) specifications of what is the expected, correct behavior. In terms of functional correctness, we often trusted the code's calculations and interactions, in the absence of any other specification. Functional correctness relative to low-level calculations (including units, scaling and quantities returned from external protocols) is generally most effectively done through thorough testing rather than human auditing.

VULNERABILITIES & FUNCTIONAL ISSUES

This section details issues affecting the functionality of the contract. Dedaub generally categorizes issues according to the following severities, but may also take other considerations into account such as impact or difficulty in exploitation:

Category	Description
CRITICAL	Can be profitably exploited by any knowledgeable third-party attacker to drain a portion of the system's or users' funds OR the contract does not function as intended and severe loss of funds may result.
HIGH	Third-party attackers or faulty functionality may block the system or cause the system or users to lose funds. Important system invariants can be violated.
MEDIUM	Examples: • User or system funds can be lost when third-party systems misbehave. • DoS, under specific conditions. • Part of the functionality becomes unusable due to a programming error.
LOW	Examples: • Breaking important system invariants but without apparent consequences. • Buggy functionality for trusted users where a workaround exists. • Security issues which may manifest when the system evolves.

Issue resolution includes "dismissed" or "acknowledged" but no action taken, by the client, or "resolved", per the auditors.

CRITICAL SEVERITY:

[NO CRITICAL SEVERITY ISSUES]

HIGH SEVERITY:

[NO HIGH SEVERITY ISSUES]

MEDIUM SEVERITY:

ID	Description	STATUS
M1	BandAdapterOracle does not check if the Arbitrum sequencer is operating	**ACKNOWLEDGED**

The BandAdapterOracle might return an invalid/stale price if the Arbitrum sequencer goes down. The Chainlink oracle that provides real time information on its function should always be checked before consuming any data.

The Stella team acknowledged the issue and is going to fix it before deploying any BandAdapterOracle contract. As of now only the ChainlinkAdapterOracle is being used.

LOW SEVERITY:

ID	Description	STATUS
L1	Liquidation status depends heavily on off-chain code	**ACKNOWLEDGED**

Function `BaseStrategy::liquidatePosition` defines the condition that must be satisfied in order for a position to be considered liquidatable:

`BaseStrategy::LiquidatePosition():L349-356`

```
if (
    pos.status != PositionStatus.ACTIVE ||
    (debtRatioE18 < ONE_E18 &&
        pos.startLiqTimestamp == 0 &&
        pos.positionDeadline > block.timestamp)
) {
    revert PositionNotLiquidatable(
        _params.posOwner, _params.posId, pos.status
    );
}
```

As one can observe, if `liquidatePosition` is called and it is true that `debtRatioE18 < ONE_E18` but at the same time it holds that `pos.startLiqTimestamp != 0`, the position can get liquidated. How could this happen:

1. `debtRatioE18` goes over 100% (`ONE_E18`)
2. the position is marked liquidatable, i.e., `pos.startLiqTimestamp` is set to a value different from `0`
3. `debtRatioE18` goes below `ONE_E18` before the position gets liquidated due to the user adding extra collateral or due to the value of borrowed/collateral assets changing.

At this point the position can get liquidated even though it is above water. At the same time, `pos.startLiqTimestamp` can only be set via `BasePositionManager::markLiquidationStatus`, a function which is intended mainly for position monitoring and liquidation bots to call. Thus, the marking and unmarking of positions depends heavily on off-chain code, meaning that there is always a chance that a previously unhealthy position, which becomes healthy, might not be marked as such and might get liquidated unfairly.

As a general security practice, we recommend avoiding the dependence on off-chain logic as much as possible. Ideally, even in an extreme scenario in which all monitoring

bots are down for a prolonged period of time, the effects on the protocol should not be unlimited.

In case of `liquidatePosition` an extreme downtime scenario could lead to liquidating a position with *arbitrarily low debt ratio* if it remains marked. This could be avoided by checking inside the function whether the conditions for unmarking the position are satisfied, namely that the debt ratio is below `unmarkLiqDebtRatioE18`.

The Stella team acknowledged the issue. Unfortunately, the use of bots for marking/unmarking positions is unavoidable. Also, the team would like to prevent the case where prices fluctuate around the 100% debt ratio, which can cause bots to mark and unmark too many times. That is why there is a different threshold, `unmarkLiqDebtRatioE18`*, for the marking and unmarking. Our final suggestion was to extend the liquidation condition by not allowing the liquidation of positions whose debt ratio is below* `unmarkLiqDebtRatioE18`.

L2	Sequencer downtime can cause large liquidation discount	**OPEN**

The time-based liquidation discount mechanism offers a 1% discount for each minute a position remains liquidatable. So if a liquidatable position has not been liquidated for 20 minutes, it will have a 20% discount (with a cap at 30%, that is 30 minutes).
In contrast to issue L2, this mechanism does not rely on running our own bot, since anyone has incentive to liquidate a position. Still, the discount logic does interpret the fact that liquidation did not happen within 20 minutes as “lack of incentive”, requiring to lower the price. As a consequence, if the sequencer stays offline for 30 minutes (which is not that unlikely to happen), then when it resumes operation all positions marked as liquidatable before the downtime will have a 30% discount.

If this risk is not acceptable, it could be avoided by requiring liquidators to “enable” the discount at regular intervals, before being allowed to use it. For instance they could be required to call a function every 5 or 10 minutes to enable the discount for the

corresponding period, and then wait a bit before using this discount. This essentially proves that the system is live, and that any lack of liquidation is due to insufficient incentives and not due to technical reasons.

CENTRALIZATION ISSUES:

It is often desirable for DeFi protocols to assume no trust in a central authority, including the protocol's owner. Even if the owner is reputable, users are more likely to engage with a protocol that guarantees no catastrophic failure even in the case the owner gets hacked/compromised. We list issues of this kind below. (These issues should be considered in the context of usage/deployment, as they are not uncommon. Several high-profile, high-value protocols have significant centralization threats.)

ID	Description	STATUS
N1	Limiting the power of dev/exec wallets	**INFO**

The dev and exec multisig wallets have significant power over the protocol as they can set all its different parameters, including access controllers, whitelisted borrowers (as mentioned in issue L1), the RiskFramework contract, the rewards vault, the oracle sources and many more. Nevertheless, the current codebase of the protocol does not provide any direct way to the owners of these wallets, even in a case of wallet compromise, to drain the protocol or steal user funds. The only exceptions to this are

1. What is described in issue L1, and
2. That the exec multisig is able transfer the whole balance of a rewards vault to the treasury, which is also controlled (set) by the exec multisig.

Although some level of trust to these wallets is necessary, limiting their power as much as possible is preferable.

OTHER / ADVISORY ISSUES:

This section details issues that are not thought to directly affect the functionality of the project, but we recommend considering them.

ID	Description	STATUS
A1	Borrower checks can be stricter	**RESOLVED**
In function `LendingProxy::borrow` it is checked that the `msg.sender` is whitelisted by querying the `whitelistedBorrowers` mapping, which can be only set by the *exec* role (multisig). However, it could also be required as an extra security measure that the `msg.sender` is a strategy registered in the `StrategyRegistry` contract, i.e., it has been created by one of the whitelisted strategy factories. This would make it much more difficult to add a non-legitimate strategy as a borrower even if the *exec* multisig got compromised.		
A2	ERC20 transfers might be of 0 value	**INFO**
The function `LendingProxy::shareProfit` does not check if the toTreasury and toRewardVault amounts used in the ERC20 safeTransferFrom calls are greater than 0.		
A3	Number of decimals in oracle's answer is hard-coded	**INFO**
In function `ChainlinkAdapterOracle::getUSDPriceE36` it is assumed that the number of decimals of the `latestRoundData` answer is 8 and 18 when the reference asset is USD and ETH respectively. Ideally one should use the decimals value returned by the `latestRoundData` function as the aforementioned assumption might not hold in the future.		
A4	Extra oracle checks	**INFO**

Function `ChainlinkAdapterOracle::getUSDPriceE36` could implement an extra check that ensures that the rate returned by the aggregator's `latestRoundData` is greater than `0`, as we expect that to hold for all asset prices. It could also be checked that the `updatedAt` value is not completely erroneous by requiring it be less or equal to the `block.timestamp`.

A5	Function can return early to save on gas	**INFO**

The function BasePositionViewer::calcProfitInfo can return early, i.e., without calculating the yield and lender profits, which would be equal to 0, when the inputShareE18 is equal to 100%.

A6	Gas optimization in case of execution reversion	**INFO**

The minDesiredHealthFactorE18 check in function getLiquidationDiscountMultiplierE18 of the BasePositionViewer contract should be moved to the start of the function to reduce the consumed gas in cases where the execution reverts due to this check.

A7	Compiler version and possible bugs	**INFO**

The code can be compiled with Solidity versions ^0.8.19. According to the foundry.toml file of the codebase, version 0.8.19 is currently used which has some known bugs, which we do not believe affect the correctness of the contracts.

DISCLAIMER

The audited contracts have been analyzed using automated techniques and extensive human inspection in accordance with state-of-the-art practices as of the date of this report. The audit makes no statements or warranties on the security of the code. On its own, it cannot be considered a sufficient assessment of the correctness of the contract. While we have conducted an analysis to the best of our ability, it is our recommendation for high-value contracts to commission several independent audits, a public bug bounty program, as well as continuous security auditing and monitoring through Dedaub Watchdog.

ABOUT DEDAUB

Dedaub offers significant security expertise combined with cutting-edge program analysis technology to secure some of the most prominent protocols in DeFi. The founders, as well as many of Dedaub's auditors, have a strong academic research background together with a real-world hacker mentality to secure code. Protocol blockchain developers hire us for our foundational analysis tools and deep expertise in program analysis, reverse engineering, DeFi exploits, cryptography and financial mathematics.