Audit Report

Neptune Protocol

v1.0

April 9, 2023

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by Cryptech Developments Ltd. to perform a security audit of Neptune Protocol.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following targets:

Repository	https://github.com/cryptechdev/neptune-protocol
Commit	`4fe9176b31cf7a47f5bb4705c981861289ef24d9`
Scope	All contracts were in scope, except the `leverage_trading` feature.
Fixes reviewed at commit	`66834f051d8750b2ef7ea917fe1c6710c0a345f7`

Repository	https://github.com/cryptechdev/neptune-auth
Commit	`c8127eab528e77f9189988e81ee0485e261c408b`
Scope	All code was in scope.
Fixes reviewed at commit	`e605c7e339e08f3811ebcf6ee06fc6eaf9f41c66`

Repository	https://github.com/cryptechdev/neptune-common
Commit	`efd313d353fafde910b9e03a9adc5c7449e3bed3`
Scope	All code was in scope.
Fixes reviewed at commit	`e883183dfc0648ec60c0d694dbb451d3cecd5d97`

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

Neptune Protocol is a decentralized lending protocol with a novel PID transformed interest rate curve.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending, Acknowledged**, or **Resolved**.

Note that audits are an important step to improving 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 the 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**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	**Medium**	Due to the novelty of the interest rate mechanism, the reliance on different off-chain components, and a large feature set (with features such as flash loans), the code base is non-trivial in several places.
Code readability and clarity	**Medium-High**	-
Level of documentation	**Medium-High**	-
Test coverage	**Medium-High**	74.76% test coverage

Summary of Findings

No	Description	Severity	Status
1	Interest rate is not related to overall lending activity	Critical	Resolved
2	Users can take out free loans	Major	Resolved
3	Lost funds when a user returns too much tokens after a flash loan	Major	Resolved
4	Updated `interest_fee` is used for whole timespan since last update	Major	Resolved
5	Accounts containing debt or collateral of an asset removed from the oracle contract are locked and not liquidatable	Major	Resolved
6	Interest distribution could run out of gas	Minor	Resolved
7	Returning funds might run out of gas	Minor	Acknowledged
8	Price oracle contract is prone to bots race condition	Minor	Acknowledged
9	Economic and risk management framework is incomplete	Minor	Acknowledged
10	Oracle centralization risks	Minor	Acknowledged
11	PID parameter centralization risk	Minor	Resolved
12	Setpoint updates are costly or cause market turmoil	Minor	Acknowledged
13	Liquidation incentives might lead to postponed liquidations and create systemic risks	Minor	Acknowledged
14	Updating the receipt token contract address of an existing market renders emitted shares not redeemable	Minor	Resolved
15	During contract ownership transfer, the new Owner address is not validated	Minor	Resolved
16	Oversent funds lead to failure loan repayment	Minor	Resolved
17	Contracts are not compliant with CW2 Migration specification	Minor	Resolved
18	Assets are added or updated in the interest model	Minor	Resolved

	and oracle contract without validating parameters		
19	`receipt_addr` is not validated	Minor	Resolved
20	Contract configurations are not validated	Minor	Resolved
21	Market contract’s `Params` are not validated	Minor	Resolved
22	Pool balance may underflow because of dust	Informational	Acknowledged
23	Generic contract configurations allow the Owner to store unnecessary information in the contract	Informational	Acknowledged
24	Contracts should implement a two step ownership transfer	Informational	Acknowledged
25	“Migrate only if newer” pattern is not followed	Informational	Acknowledged
26	Invalid logo for multiple tokens	Informational	Acknowledged

Detailed Findings

1. Interest rate is not related to overall lending activity

Severity: Critical

When the borrow rate is queried in `contracts/market/src/execute.rs:659`, the rate is calculated only on the basis of the current utilization. Thus, the rate is not related to the average lending and borrowing activity since the last call but reflects only the current utilization. This might lead to scenarios where users do not receive interest rates (because the rate is very low in the current block) although they lent out capital for a substantial period of time when interest rates were higher. Similarly, borrowers might pay extremely high interest rates in the opposite scenario.

An attacker could exploit this issue by withdrawing ample liquidity for only one block such that unaware borrowers are forced into liquidations due to the high interest rates. Equivalently, long-term borrowers can manipulate interest rates down for one block in order to pay very low interest rates. Both attack vectors only work probabilistically, as the interest payment is triggered randomly, with an equal probability for all blocks. The attack would therefore be risky and require the right preconditions. Still, while the likelihood of a successful attack low, the impact is very high, so we classify this issue as critical.

Recommendation

We recommend calculating the interest rate such that it reflects the average utilization since the last call.

Status: Resolved

2. Users can take out free loans

Severity: Major

There is no logic implemented within the `execute_borrow` or `execute_return` functions to ensure users pay interest for loans. Consequently, users may borrow within short time frames without paying fees, if `execute_distribute_interest` is not called during their borrowing period. This could happen within one block but also across multiple blocks.

This issue incentivizes users to overborrow, which puts the funds of lenders at risk if bad debt accumulates. For example, arbitrageurs might use large amounts of capital in volatile markets for free until they default. In addition, this issue allows users to circumvent the flash loan fee.

Recommendation

We recommend distributing interest whenever a user borrows or returns funds or implementing another mechanism that enforces the payment of interest rates for all users.

Status: Resolved

3. Lost funds when a user returns more tokens than they borrowed with a flash loan

Severity: Major

In `contracts/market/src/execute.rs:120`, the `saturating_sub` function is used to deduct the received amount from the flash loan principal. When the received amount is higher than the principal, it will therefore be set to 0. However, the difference is not transferred back to the user, resulting in a loss of funds.

We do not classify this issue as critical despite the potential loss of funds since users do cause the loss of funds themselves by overpaying.

Recommendation

We recommend requiring an exact match (like in the `execute_return` function) or transferring the difference back to the user.

Status: Resolved

4. Updated `interest_fee` is used for whole timespan since last update

Severity: Major

The `execute_update_market_asset` function defined in `contracts/market/src/execute.rs:136-151` can be used to update the `interest_fee`. The new interest fee will then be used in `execute_distribute_interest` for the whole timespan since the last update. This can lead to situations where users pay a higher or lower protocol fee than they anticipated when they opened the position.

Recommendation

We recommend calling `execute_distribute_interest` before the `interest_fee` is updated.

Status: Resolved

5. Accounts containing debt or collateral of an asset removed from the oracle contract are locked and not liquidatable

Severity: Major

The `query_and_validate_prices` function defined in `packages/neptune-money-market/src/queries.rs:42-53`, which is during the handling of `Borrow`, `WithdrawCollateral` and `Liquidate` messages, interacts with the price oracle contract in order to get asset prices by executing the `GetPrices` query.

Since this query returns an `AssetNotFound` error if one of the requested assets has been deleted from the oracle contract, all the mentioned messages interacting with an account that contains a removed asset in its `debt_pool_accounts` and `collateral_pool_accounts` attributes will fail.

Consequently, affected accounts cannot borrow, withdraw collateral, or be liquidated.

Recommendation

We recommend handling the error or not returning an error when querying the price of removed assets.

Status: Resolved

6. Interest distribution could run out of gas

Severity: Minor

The `execute_distribute_interest` function defined in `contracts/market/src/execute.rs:647-701` iterates through all the existing markets in order to distribute interests and accrue fees in the war chest.

Since it is iterating through all the markets and for each of them a smart query and a message are executed, the execution could run out of gas if the market's cardinality is considerable.

Consequently, interest may not be distributed anymore.

Recommendation

We recommend implementing a mechanism for distributing interests in batches to a subset of markets or limiting the maximum number of markets.

Status: Resolved

7. Returning funds might run out of gas

Severity: Minor

The `execute_return` function might run out of gas in `contracts/market/src/execute.rs:372` as it iterates through all assets in order to save and prune the account. These iterations might run out of gas if too many assets are in the account.

Consequently, it may be impossible to return funds.

We are reporting this issue with minor severity since it only occurs when the admin allows a large number of assets in the contract.

Recommendation

We recommend limiting the number of assets per account.

Status: Acknowledged

8. Price oracle contract is prone to bots race condition

Severity: Minor

The price oracle contract allows `Owner` and `Bots` to publish asset prices using the `execute_update_prices` function defined in `contracts/price_oracle/src/contract.rs:56-77`. However, since multiple `Bots` can submit their price in the same block and the oracle contract stores only one price for each asset, race conditions are possible where the latest transaction overwrites all the other bots' submitted data.

Recommendation

We recommend aggregating prices from multiple bots rather than overwriting them. For example, the medium price could be calculated per block, in order to make the oracle more robust to compromised bots.

Status: Acknowledged

9. Economic and risk management framework is incomplete

Severity: Minor

The economic and risk management framework is incomplete. It is not stated what the key economic outcomes for the users are. The assumptions used in the stress-testing simulations violate common economic assumptions and the stress-testing setup is not fully documented. The tuning risk might not be addressable by data collection. The incompleteness is largely related to the manifold of possible parametrizations of the PID controller and the lack of elaborations or investigations into how the stabilized utilization affects the risk-return profiles of users.

The Appendix elaborates on this issue extensively.

Eventually, parameterizations or updates of parameterizations that prioritize stable utilization over micro- and macroprudential risks might put a significant amount of users' funds at unassessed risks.

Recommendation

We recommend re-iterating the risk management framework and communication. The parameters of the model should be stated explicitly. If they are subject to change also future parameters should be disclosed. We recommend assessing the PID stabilization of the utilization rate with a risk-return perspective and elaborating on what happens under conventional economic assumptions.
We encourage clarifying why and how risk and return metrics are better than a given benchmark.
We also recommend implementing a stress testing framework like Aave’s Market Risk Assessment which adapted approaches from banking stress testing (namely the Fed and ECB frameworks) for DeFi.

Status: Acknowledged

10. Oracle centralization risks

Severity: Minor

In the current design, the owner and bots can post arbitrary prices to the oracle. Prices are not validated and every value is accepted. This can be problematic for multiple reasons:

1. Bugs in the off-chain bots: A bot may have a bug (or use a data source that delivers wrong data), leading to wrong updates.
2. Privilege abuse: An attacker may steal the private key of a bot (e.g., by attacking the server of the bot). He can then use the bot’s account to manipulate prices, for example by setting the price for all assets to 0, which would allow the attacker to liquidate all users at their loss.
3. Market pause: Since the market checks prices’ timestamp in `packages/neptune-money-market/src/queries.rs:46-52` and does not perform operations on expired data, bots malfunctioning could lead to market operations being delayed or paused. This would also affect liquidations.

Recommendation

We recommend performing validation on the posted prices. For instance, there could be a maximum allowed delta per time unit, such that a price of 0 would not be accepted. While this does not fully resolve the centralization issue, privilege abuse would be more involved (and require multiple transactions over a longer timespan). This would allow operators and users to react. Furthermore, bugs in the off-chain bots that result in clearly wrong prices would no longer break the whole system. Alternatively, prices could be aggregated, for example by calculating a time-weighted median price to minimize the impact of a single buggy or compromised bot.

Status: Acknowledged

11. PID parameter centralization risk

Severity: Minor

In the current design, the owner can reset arbitrary PID parameter values in an unlimited manner which is equivalent to control over lending rates and prices. Similar to the issue Oracle centralization risks, this has security implications as the owner could make mistakes while setting parameters or a compromised owner account could be used to exploit the protocol.

Recommendation

We recommend limiting parameter updates in both time and value dimensions. For example, allowing an update only once per week and limiting the relative change to a maximum of 10%.

Status: Resolved

12. Setpoint updates are costly or cause market turmoil

Severity: Minor

If a setpoint is updated close to the steady state the error grows as defined in formula 11 of the whitepaper. Consequently, the PID controller will react by adjusting interest rates more aggressively. For larger set point updates this might cause market turmoil, as rates hike or plunge. To circumvent such volatility, the admin or a bot could incrementally update the set point over multiple calls – but this is costly and error prone and therefore a violation of best practices.

Recommendation

We recommend implementing a setpoint reset schedule such that set points are updated successively either as a function of time and/or as a function of the squared error (update only if the squared error is within a tolerance, and smaller updates if the current error is large and would become larger).

Status: Acknowledged

13. Liquidation incentives might lead to postponed liquidations and create systemic risks

Severity: Minor

Equation 6 in the whitepaper describes how the discount factor is inversely proportional to the health of the account. The discount is higher, if the health is lower. Since a larger discount equals a larger payout, this sets the incentive for liquidators to liquidate later if they anticipate price movements (e.g. when prices of collateral are trending downwards) that deteriorate some accounts' health, which are subject to liquidation. Consequently, liquidation gains a speculative aspect in addition to the risk-management aspect. This increases the risk of the protocol, keeping all else equal. If downward trends are pronounced, this might put the whole protocol at the risk of accumulating bad debt, because liquidations happen too late due to speculative behavior.

Recommendation

We recommend aligning the incentives of liquidators strictly with risk management by making discounts independent of the health of the account. Alternatively, a parametrization could be used that calculates the systemic default contribution of an individual account – e.g. inspired by financial connectedness measures or the CoVAR methodology – and provide additional rewards to liquidators who liquidate accounts with greater systemic risk contributions.

Status: Acknowledged

The client replied that the discount will be set to very small values between 0.01-0.02 to minimize the impact of this issue. Furthermore, the client stated: "This means that liquidations will be maximally profitable at an account health of 0.98-0.99. Assuming the worst-case scenario that liquidators only ever choose to liquidate at a maximum discount, this would be equivalent to setting the liquidation LTV of each asset approximately 1-2% higher. What this means is that any systemic risk imposed by the dynamic liquidation discount can be fully accounted for by decreasing the liquidation LTV of a particular asset by the corresponding amount."

14. Updating the receipt token contract address of an existing market renders emitted shares not redeemable

Severity: Minor

The `execute_update_market_asset` function defined in `contracts/market/src/execute.rs` allows the `Owner` to update a `Market` instance including its relative receipt token contract address in `RECEIPT_ORIGIN`.

Since users might hold shares of a market that is updated, a substitution of the receipt token contract address would render those shares not redeemable.

We classify this issue as minor since only the owner can cause it.

Recommendation

We recommend not allowing updates of the receipt token contract address while shares are outstanding.

Status: Resolved

15. During contract ownership transfer, the new `Owner` address is not validated

Severity: Minor

During contract ownership transfer, the `set_config` function defined in `packages/neptune-money-market/src/config.rs:122-130` does not validate the new `Owner`'s address.

Consequently, in case of a typo in the new `Owner`'s address, the contract will not be administrable anymore and protocol critical messages can no longer be executed.

We classify this issue as minor since only the owner can cause it.

Recommendation

We recommend validating the new `Owner`'s address before storing it.

Status: Resolved

16. Oversent funds lead to failure loan repayment

Severity: Minor

In contracts/market/src/execute.rs:359-361, the execute_return function fails when the user returns too many funds, such that the user's position cannot be closed. This could be avoided by sending additional funds back to the user.

Recommendation

We recommend sending the additional funds back to the user.

Status: Resolved

17. Contracts are not compliant with CW2 Migration specification

Severity: Minor

The contracts in scope of this audit do not adhere to the CW2 Migration specification standard.

This may lead to unexpected problems during contract migration and code version handling.

Recommendation

We recommend following the CW2 standard in all the contracts. For reference, see https://docs.cosmwasm.com/docs/1.0/smart-contracts/migration.

Status: Resolved

18. Assets are added or updated in the interest model and oracle contract without validating parameters

Severity: Minor

The execute_add_asset and execute_update_asset functions defined in contracts/interest_model/src/contract.rs:110-118, contracts/price_oracle/src/contract.rs:91-99 and contracts/interest_model/src/contract.rs:129-156, do not validate an asset's parameters when storing or updating it.

Specifically, the address wrapped in the AssetInfo struct is not validated and the time_last_updated timestamp can incorrectly contain a value.

Additionally, in the interest model contract, the value of sample_period_min should be enforced to be less than the value of sample_period_max and curve should be at least a non-empty vector but might also require more numerical validation to ensure that parameter updates cannot lead to instability of the PID controller. Similarly, all `pid_params` and `pid_state` should be numerically validated to be within reasonable boundaries – for example, the setpoint should be greater than zero and less than one.

Recommendation

We recommend performing validation on asset parameters before storing or updating them.

Status: Resolved

19. `receipt_addr` is not validated

Severity: Minor

The `receipt_addr` in `contracts/market/src/execute.rs:139` is not validated.

Recommendation

We recommend validating addresses.

Status: Resolved

20. Contract configurations are not validated

Severity: Minor

The `set_config` function defined in `packages/neptune-money-market/src/config.rs:122-130` stores the configuration provided by the `Owner` without performing validation.

Addresses provided in both the `ConfigMsg`'s `uniques` and `vecs` attributes are not validated.

Additionally, since elements in `vecs`'s vectors are stored iteratively, they should be checked to not contain duplicates in order to avoid overwriting.

Recommendation

We recommend validating addresses and deduplicating vectors.

Status: Resolved

21. Market contract’s `Params` are not validated

Severity: Minor

In `contracts/market/src/contract.rs:29`, during the market contract’s instantiation and in `contracts/market/src/execute.rs:45-50`, provided `Params` are not validated. For example, `liquidation_fee` and `flash_loan_fee` attributes should be validated to be in the `[0,1]` range and the address in `hub_asset` should be valid.

Also in `contracts/market/src/execute.rs:173`, the input addresses of `LIQUIDITY_POOLS` are not validated and `assets[0]` is not checked to be different from `assets[1]`.

Recommendation

We recommend validating `Params` before storing them in the contract, validating the addresses, and ensuring that assets are not equal.

Status: Resolved

22. Pool balance may underflow because of dust

Severity: Informational

In `src/pool.rs:152`, `saturating_sub` is used for removing the amount from the `account_principal`. However, in line `155`, a normal subtraction is used, where the value may be different from the one that was subtracted from `account_principal` (when `amount_to_remove > account_principal`). While the difference should be small due to the calculations and checks above, that small difference between the sum of all `account_principal` values and the `pool_balance` variable will accrue over time. This can lead to a situation where the last withdrawal underflows if the user tries to withdraw their whole principal.

Recommendation

We recommend decrementing `pool_balance` by the same amount as `account_principal`.

Status: Acknowledged

23. Generic contract configurations allow the Owner to store unnecessary information in the contract

Severity: Informational

The `SetConfig` message defined in all contracts uses generic logic in order to update contract configurations without taking into account specifics.

For example, it is possible for an `Owner` to set the `WarChest` or the `PriceOracle` addresses in the configurations of the oracle contract which does not require those configurations.

Recommendation

We recommend defining specific contract configuration structures and setters.

Status: Acknowledged

24. Contracts should implement a two-step ownership transfer

Severity: Informational

The contracts within the scope of this audit allow the current owner to execute a one-step ownership transfer. While this is common practice, it presents a risk for the ownership of the contract to become lost if the owner transfers ownership to the incorrect address. A two-step ownership transfer will allow the current owner to propose a new owner, and then the account that is proposed as the new owner may call a function that will allow them to claim ownership and actually execute the config update.

Recommendation

We recommend implementing a two-step ownership transfer. The flow can be as follows:

1. The current owner proposes a new owner address that is validated and lowercased.
2. The new owner account claims ownership, which applies the configuration changes.

Status: Acknowledged

25. “Migrate only if newer” pattern is not followed

Severity: Informational

The contracts within the scope of this audit are currently migrated without regard to their version. This can be improved by adding validation to ensure that the migration is only performed if the supplied version is newer.

Recommendation

We recommend to follow the migrate "only if newer" pattern defined in the CosmWasm documentation.

Status: Acknowledged

26. Invalid logo for multiple tokens

Severity: Informational

In `packages/defauts/src/noria.rs`, the logo for multiple tokens (`nNORIA`, `nCRD`, `nNEPT`) is set to the placeholder URL `http://acme.com`.

Recommendation

We recommend replacing these placeholder URLs with the correct ones.

Status: Acknowledged

Appendix: Assessment of the economic and risk management framework

Intuition

The whitepaper states that a stable utilization is desirable, but does not elaborate why. A more stable utilization could result in various economic outcomes such as lower/higher interest rate volatility (depending on the PID parameters), higher/lower systemic risk (because utilization and borrowing are (dis-)incentivized), higher/lower liquidation likelihood, higher\lower borrowing rates and lending rates (borrowers pay more/less risk or liquidity premia than in other markets).

Any of those might be possible under certain parameterizations and most of these have strong implications for the funds of users vis-a-vis traditional lending curves. Fixing (as an extreme case) or controlling (in the case of PID controller) one of the degrees of freedom of an equation system or market (here the utilization rate of capital) will always have an impact on the pricing of assets. Figure 2 highlights that there is not only a steady-state utilization rate, but also a steady-state borrowing and lending rate. These steady-state values can be de facto set by the parameter values. Consequently, it is also of interest to users how the protocol will work and behave under the presence of other lending protocols that have initially more liquidity and diverging interest rates.

Stress testing assumptions and key metrics:

The whitepaper is focused on utilization rates as an outcome variable. In finance and risk management applications such indicators usually enter risk-management frameworks as an input variable, and the output variables are typically measures of risk, return, or risk-return tradeoffs.

- On the individual level key metrics to measure individual risk and returns include:
  - For borrowers:
    - Price of capital
    - Risk of interest rate changes
    - Risk of liquidation
  - For lenders:
    - Return on assets
    - Risk of interest rate changes (as an income risk)
    - Risk of defaults (partial, full)
    - Income volatility
  - For liquidators:

        - Incentives (return) on liquidations
- On the aggregate level, risk measures typically include:
    - Likelihood of bankruptcy
    - Liquidator compliance
    - Likelihood of mass liquidations, for example, driven by
        - Risk of collateral-deleveraging spirals – price changes of collateral that require a large sale of collateral to stabilize LTV ratios, but trigger lower prices such that more collateral is required to be sold. Similarly, this may be triggered by liquidations of accounts with low-quality collateral, which is then sold off
        - Collateral correlation risks

The stress testing perturbations in section 7.5 are a good tool to assess the ability to bring the utilization rate to the desired level and could also be used to investigate the model along the risk dimension. However, the current approach deviates from best-practice modeling approaches used in economics, where typically forward-looking agents with rational expectations, no-arbitrage conditions, and benchmarking are used.

The model assumes static and benevolent agents – this means that they are not trying to maximize/optimize profits or behave in a strategic manner and are agnostic about the future. In economics, one typically assumes agents that behave strategically, employ rent-seeking and have a (sometimes limited) model of the future. In finance or macroeconomics, these agents are typically modeled along the lines of an agent with a given degree of risk aversion, time preference, and income/consumption maximizing behavior. In game theory, outcomes are modeled mostly directly in terms of utility. However, there are also other and more exotic approaches where agents are modeled as PID controllers. In any of the cases, the agents are not static but *adaptive* to their environment. The agent takes the market conditions and rules into account while making decisions. In the given protocol, this could mean that agents could try to predict future PID errors (in terms of equation 11 in the whitepaper) and capitalize on the interest changes introduced by the PID controller. Agents could also try to create those errors in order to trigger a certain controller response.

This is particularly possible if agents have access to additional capital, which leads to another typical condition in economics, the no-arbitrage condition. In theoretical finance, one typically assumes that the same assets (along the risk-liquidity-return dimensions) should have the same price across markets in a steady state. The whitepaper is partially agnostic of outside capital prices, volumes, and risk premia. While it discusses changes in borrowing, lending amounts, and price, it is not investigated if there is a second market (such as for example Compound or Aave), that offers capital without a utilization constraint and possibly diverging interest rates.

The third point is benchmarking: Figure 2 depicts how the algorithm converges to a steady state after 35000 (x-axis undocumented). The figure is not self-contained, as it is not stated why this value is good or desirable and if and how it outperforms other models.

Tuning Risk

As a further point, we re-emphasize the tuning risk, which is already highlighted in the whitepaper. Besides the data limitations mentioned in the whitepaper, the learning nature of users is a risk. Malicious users can track and learn about the algorithm on-chain and gather empirical data about the protocol. Such observant users might not enter the market (and remain invisible in the data) until a parameterization is set which can be exploited. This risk can only be addressed by rigorous risk modeling.