Zellic

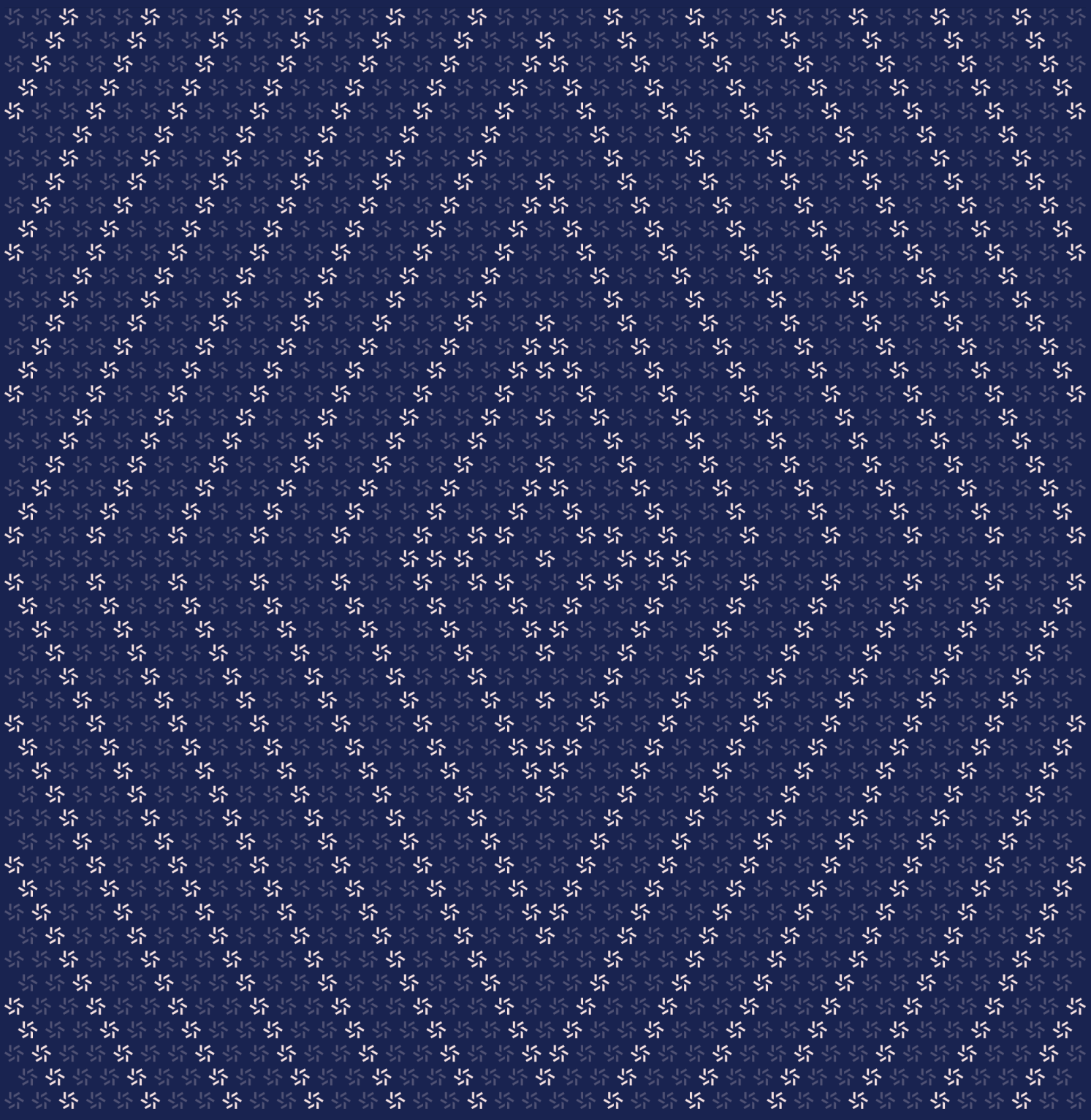
Prepared for
Tyler Loewen
Adrastia

Prepared by
Jaeeu Kim
Seungjun Kim
Zellic

May 9, 2024

Adrastia PID Controller

Smart Contract Security Assessment



Contents

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for Adrastia from April 29th to May 3rd, 2024. During this engagement, Zellic reviewed Adrastia PID Controller's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Could an on-chain attacker change the rate values or configs in PID controller?
- Could an on-chain attacker make the contract behave in an unintended manner to deny service to users?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody
- Any off-chain components that perform validation checks on price or liquidity data

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

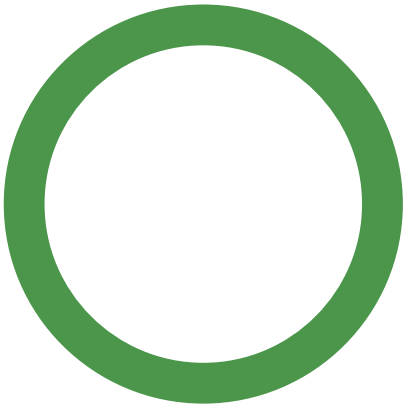
1.4. Results

During our assessment on the scoped Adrastia PID Controller contracts, we discovered two findings, both of which were low impact.

Additionally, Zellic recorded its notes and observations from the assessment for Adrastia's benefit in the Discussion section (4. ↗).

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	0
Low	2
Informational	0



2. Introduction

2.1. About Adrastia PID Controller

Adrastia contributed the following description of Adrastia PID Controller:

> Adrastia is a blockchain data and automation platform that increases the economic and technical security of DeFi platforms through robust oracles and data controllers.

> Adrastia's PID controller system automatically optimizes the output to minimize the input error. The primary use case is managing interest rates — input is a pool utilization rate, the error is the difference between the utilization rate and the target, and the output is the interest rate. It updates upon a fixed period and reads from Adrastia's Time-Weighted Value and Error Oracles whose averaging period matches the controller period.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We

also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped contracts itself. These observations — found in the Discussion (4. ↗) section of the document — may include suggestions for improving the codebase, or general recommendations, but do not necessarily convey that we suggest a code change.

2.3. Scope

The engagement involved a review of the following targets:

Adrastia PID Controller Contracts

Repositories	https://github.com/adrastia-oracle/adrastia-core ↗ https://github.com/adrastia-oracle/adrastia-periphery ↗
Versions	adrastia-core: `ea1b24be01c0467d2f8d4f49eeaa01c8108712c8` adrastia-periphery: `e41eba4b5016da4b37cd7381784722dfe2cc6c16`
Programs	• AlocUtilizationAndErrorAccumulator.sol • ValueAndErrorAccumulator.sol • ManagedAlocUtilizationAndErrorAccumulator.sol • PidController.sol • ManagedPidController.sol • TrueFiAlocPidController.sol • NegativeErrorScalingTransformer.sol • PositiveErrorScalingTransformer.sol
Type	Solidity
Platform	EVM-compatible

2.4. Project Overview

Zellic was contracted to perform a security assessment with two consultants for a total of two person-weeks. The assessment was conducted over the course of 7.5 calendar days.

Contact Information

The following project manager was associated with the engagement:

Chad McDonald
Engagement Manager
chad@zellic.io ↗

The following consultants were engaged to conduct the assessment:

Jaeeu Kim
Engineer
jaeeu@zellic.io ↗

Seungjun Kim
Engineer
seungjun@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

April 29, 2024	Start of primary review period
April 30, 2024	Kick-off call
May 3, 2024	End of primary review period

3. Detailed Findings

3.1. Admin could set PID controller's rate manually

Target	TrueFiAlocPidController, ManagedPidController, PidController		
Category	Business Logic	**Severity**	Medium
Likelihood	Low	**Impact**	Low

Description

The function `_manuallyPushRate` is used to manually set the rate of a token in the PID controller by the admin. This function could set the rates and iTerm value in the controller without any validation logic, like clamps.

```
function _manuallyPushRate(address token, uint64 target, uint64 current,
    uint256 amount) internal virtual override {
    super._manuallyPushRate(token, target, current, amount);

    // Reinitialize the PID controller to avoid a large jump in the output rate
    with the next update.
    initializePid(token);
}

function initializePid(address token) internal virtual {
    PidState storage pidState = pidData[token].state;
    BufferMetadata storage meta = rateBufferMetadata[token];

    (int256 input, int256 err) = getSignedInputAndError(token);

    pidState.lastInput = input;
    pidState.lastError = err;
    if (meta.size > 0) {
        // We have a past rate, so set the iTerm to it to avoid a large jump.
        // We don't need to clamp this because the last rate was already
    clamped.
        pidState.iTerm = int256(uint256(getLatestRate(token).current));
    }
}
```

Impact

The admin has the ability to update rates, and the next rate calculation relies on iTerm. There is a risk that the admin could inadvertently adjust the rate of the future with an incorrect iTerm.

Recommendations

Ensure that users who use this contract acknowledge and accept the risks associated with manually setting the rate.

Remediation

This issue has been acknowledged by Adrastia.

Adrastia provided the following response to this finding:

> Note that it's intended that Adrastia clients will be the ones with the sole privilege to manually push rates and set the config. It's ultimately up to them if they want to make use of this function, or if they want to adjust the permissioning of the system.
>
> If so, the Adrastia web app has multi-stage UI dialogs to assist administrators, with formatting and validations to reduce the likeliness of human error.

3.2. Underflow could occur in calculation

Target	ManagedAlocUtilizationAndErrorAccumulator, AlocUtilizationAndErrorAccumulator, ValueAndErrorAccumulator		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	Low	**Impact**	Low

Description

The function `fetchLiquidity` is used to calculate value and error in the accumulator. The function could underflow if `value - target` is greater than `ERROR_ZERO`.

```
function fetchLiquidity(bytes memory data)
    internal view virtual override returns (uint112 value, uint112 err) {
    value = fetchValue(data);
    uint256 target = fetchTarget(data);

    if (target >= value) {
        err = (ERROR_ZERO + (target - value)).toUint112();
    } else {
        err = (ERROR_ZERO - (value - target)).toUint112();
    }
}
```

Impact

If `value - target` is greater than `ERROR_ZERO`, `fetchLiquidity` is reverted due to underflow protection implemented in solidity 8.0.0. This could cause the contract to not work as expected.

Recommendations

Consider adding a check to ensure that `value - target` is less than `ERROR_ZERO`.

Remediation

This issue has been acknowledged by Adrastia.

According to Adrastia, halting updates in this scenario is considered acceptable behavior. In the

context of TrueFi's ALOCs, a reversion is appropriate since it would indicate a bug in the ALOC's `utilization` function.

4. Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security related and do not convey that we are suggesting a code change.

4.1. Setting proper configs for PID

Configuring PID controllers correctly can be challenging, and it's important to ensure that the PID controller is properly configured to prevent potential issues. The Kp, Ki, and Kd values must be set properly to ensure that the PID controller operates as intended.

This issue has been acknowledged by Adrastia.

According to the Adrastia, they have private tools set up to approximate the best values with the help of simulations.

5. Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the contracts and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

5.1. Module: AlocUtilizationAndErrorAccumulator.sol

Function: `fetchTarget(bytes)`

This is the function to get the target that is saved when this contract is constructed.

Function: `fetchValue(bytes data)`

This function is to fetch the value from the token address.

Inputs

- `data`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: The data for getting the value from the token address.

Function: `getTarget(address token)`

This is an external function to get the target by invoking the internal function `fetchTarget`.

Inputs

- `token`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: The token address for getting value from functions.

Intended branches

- Invokes the internal function `fetchTarget` with the encoded token.
  - ☑ Test coverage

Function: `liquidityDecimals()`

This function is to return the value `_liquidityDecimals` that was set in the constructor.

Function: `quoteTokenDecimals()`

This function is to return the value `_liquidityDecimals` that was set in the constructor.

5.2. Module: ValueAndErrorAccumulator.sol

Function: `fetchLiquidity(bytes data)`

This function is to fetch liquidity from the token address.

Inputs

- `data`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: The data for getting value from the token address.

5.3. Module: ManagedAlocUtilizationAndErrorAccumulator.sol

Function: `canUpdate(bytes data)`

This function is used to check if the caller can update the accumulator with the given data. Only accounts with the `ORACLE_UPDATER` role can call this function, or the role is unassigned.

Inputs

- `data`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of the data — address of the token is included.

Branches and code coverage

Intended branches

- Returns false if the message sender is missing the required role.
  - ☑ Test coverage
- Invokes the parent `canUpdate` function and returns the result.

- ☑ Test coverage

Function: `fetchTarget(bytes data)`

This function is used to fetch the target of the token. If the target is not initialized, it uses the default target.

Inputs

- `data`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Data to be decoded — address of token is included.

Branches and code coverage

Intended branches

- Decodes the data to get the token address.
  - ☑ Test coverage
- Returns the target of the token.
  - ☑ Test coverage

Function: `initializeDefaultTarget(uint112 target)`

This function is used to initialize the default target.

Inputs

- `target`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of the default target.

Branches and code coverage

Intended branches

- Updates the target for the default target and sets the initialized flag to true.
  - ☑ Test coverage

Function: `isUsingDefaultTarget(address token)`

This function is used to check if the given token is using the default target. It returns false when the target of the token is initialized; otherwise, it returns true.

Inputs

- `token`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the token.

Branches and code coverage

Intended branches

- Returns true if the token is `address(0)`.
  - ☑ Test coverage
- Returns false if the target of the token is initialized.
  - ☑ Test coverage
- Returns true if the target of the token is not initialized.
  - ☑ Test coverage

Function: `revertToDefaultTarget(address token)`

This function is used to revert the target for the given token to the default target. Sets the initialized flag to false.

Inputs

- `token`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the token.

Branches and code coverage

Intended branches

- Sets the initialized flag to false.
  - ☑ Test coverage

Negative behavior

- Reverts if the token address is zero.
  - ☑ Negative test
- Reverts if the token is not initialized.
  - ☑ Negative test

Function: `setTarget(address token, uint112 target)`

This function is used to set a new target for the given token's config.

Inputs

- `token`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the token.
- `target`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of the target.

Branches and code coverage

Intended branches

- Updates the target for the given token's config and sets the initialized flag to true.
  - ☑ Test coverage

Negative behavior

- Reverts if the target is already initialized and the new target is the same as the current target.
  - ☑ Negative test

Function: `supportsInterface(byte[4] interfaceId)`

This function is used to check if the contract supports the given interface.

Inputs

- `interfaceId`
  - **Control**: Arbitrary.
  - **Constraints**: None.

- **Impact**: Interface ID to check.

Branches and code coverage

Intended branches

- Invokes the parent `supportsInterface` functions to check if the contract supports the given interface.
  - ☑ Test coverage
- Returns true if the contract supports the given interface.
  - ☑ Test coverage

Function: `update(bytes data)`

This function is used to update the accumulator with the given data. Only accounts with the `ORACLE_UPDATER` role can call this function.

Inputs

- `data`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of the data — address of the token is included.

Branches and code coverage

Intended branches

- Invokes the parent `update` function.
  - ☑ Test coverage

Negative behavior

- Reverts if the caller is not an oracle updater or the oracle updater role is not open.
  - ☑ Negative test

Function: `_heartbeat()`

This function is used to return the heartbeat of config.

Branches and code coverage

Intended branches

- Returns the heartbeat of config.
  - ☑ Test coverage

Function: `_updateDelay()`

This function is used to return the update delay of config.

Branches and code coverage

Intended branches

- Returns the update delay of config.
  - ☑ Test coverage

Function: `_updateThreshold()`

This function is used to return the update threshold of config.

Branches and code coverage

Intended branches

- Returns the update threshold of config.
  - ☑ Test coverage

5.4. Module: ManagedPidController.sol

Function: `canUpdate(bytes data)`

This function is used to check if the sender can update the rates.

Inputs

- `data`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Data to be decoded — address of token is included.

Branches and code coverage

Intended branches

- Invoke the parent contract's `canUpdate` function.
  - ☑ Test coverage
- Returns true if the caller is an oracle updater or the oracle updater role is open and the parent contract's `canUpdate` function returns true.
  - ☑ Test coverage

Function: `initializeRoles()`

This function is used to initialize the roles' hierarchy. It is called in the constructor.

Branches and code coverage

Intended branches

- Sets up the admin role, setting `msg.sender` as the admin.
  - ☑ Test coverage

Function: `supportsInterface(byte[4] interfaceId)`

This function is used to check if the contract supports the interface with the given `interfaceId`.

Inputs

- `interfaceId`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value to be checked if the contract supports the interface.

Branches and code coverage

Intended branches

- Invoke the parent contract's `supportsInterface` function and returns the result.
  - ☑ Test coverage

5.5. Module: NegativeErrorScalingTransformer.sol

Function: `transformInputAndError(int256 input, int256 error)`

This function is used to transform error value as per the scaling factor when the error is negative.

Inputs

- `input`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of input.
- `error`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of error to be transformed.

Branches and code coverage

Intended branches

- Scales the error if it is negative.
  - ☑ Test coverage
- Returns the unchanged input value and transformed error values.
  - ☑ Test coverage

5.6. Module: PidController.sol

Function: `checkSetDefaultInputAndErrorOracle()`

This function is used to check if the caller is authorized to set the default input and error oracle.

Branches and code coverage

Intended branches

- Invokes the internal function `checkSetConfig`.
  - ☑ Test coverage

Function: `checkSetPidConfig()`

This function is used to check if the caller is authorized to set the PID configuration.

Branches and code coverage

Intended branches

- Invokes the internal function `checkSetConfig`.
  - ☑ Test coverage

Function: `clampBigSignedRate(address token, int256 value, bool isOutput, bool clampChange, int256 last)`

This function is used to clamp a big signed rate as per the token's configuration.

Inputs

- `token`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the token.
- `value`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value to clamp.
- `isOutput`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Flag to indicate if the value is an output rate.
- `clampChange`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Flag to indicate if the change should be clamped. Only relevant if `isOutput` is false.
- `last`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Last value for comparison. Only relevant if `isOutput` is false.

Branches and code coverage

Intended branches

- Sets `valueAsUint` as `type(uint64).max` if the value is greater than or equal to `type(uint64).max`.
  - ☑ Test coverage
- Sets `valueAsUint` as zero if the value is less than zero.
  - ☑ Test coverage
- Sets `valueAsUint` as `uint64(uint256(value))` if the value is between zero and `type(uint64).max`.
  - ☑ Test coverage
- Invokes the internal function `clamp` if `isOutput` is true with the token and `valueAsUint` and returns the clamped value.

☑ Test coverage

- If `isOutput` is false, uses the `last` to get `lastAsUint`.
  ☑ Test coverage
- Sets `lastAsUint` as zero if the `last` is less than or equal to zero.
  ☑ Test coverage
- Sets `lastAsUint` as `type(uint64).max` if the `last` is greater than or equal to `type(uint64).max`.
  ☑ Test coverage
- Sets `lastAsUint` as `uint64(uint256(last))` if the `last` is between zero and `type(uint64).max`.
  ☑ Test coverage
- Invokes the internal function `clampWrtLast` if `isOutput` is false with the `token`, `valueAsUint`, `clampChange`, and `lastAsUint` and returns the clamped value.
  ☑ Test coverage

Function: `computeNextPidRate(address token)`

This function is used to compute the next PID rate for a given token.

Inputs

- `token`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the token.

Branches and code coverage

Intended branches

- Gets delta time if the buffer size is greater than zero.
  ☑ Test coverage
- Invokes the internal function `getSignedInputAndError` to get the signed input and error values.
  ☑ Test coverage
- Gets pTerm using the PID configuration and input or error values.
  ☑ Test coverage
- Gets iTerm using the PID configuration by invoking the internal function `clampBigSignedRate` with the token and old iTerm.
  ☑ Test coverage
- Gets dTerm if delta time is greater than zero.
  ☑ Test coverage
- Calculates the output by adding pTerm, iTerm, and dTerm using the PID algorithm.

☑ Test coverage

- Clamps the output using the internal function `clampBigSignedRate` with the token and output.
  ☑ Test coverage
- Returns target and the PID state updated with the last input, last error, and output.
  ☑ Test coverage

Negative behavior

- Reverts if the rate config of token has not been set yet.
  ☑ Negative test
- Reverts if the `kPDenominator` of PID config is zero.
  ☑ Negative test

Function: `computeRateInternal(address token)`

This function is used to compute the rate for a given token and returns the target rate.

Inputs

- `token`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the token.

Branches and code coverage

Intended branches

- Invokes the internal function `computeNextPidRate` with the token and returns the target rate.
  ☑ Test coverage
- Returns the target from the internal function `computeNextPidRate`.
  ☑ Test coverage

Function: `computeRate(address token)`

This function is used to return the current rate for the token.

Inputs

- `token`
  - **Control**: Arbitrary.

- **Constraints**: None.
- **Impact**: Address of the token.

Branches and code coverage

Intended branches

- Invokes the internal function `getLatestRate` with the token.
  - ☑ Test coverage
- Returns the current rate of the token.
  - ☑ Test coverage

Negative behavior

- Reverts if the rate has never been computed.
  - ☑ Negative test

Function: `getInputAndErrorOracle(address token)`

This function is used to get the address of the input and error oracle for a given token.

Inputs

- `token`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the token.

Branches and code coverage

Intended branches

- Invokes the internal function `_inputAndErrorOracle`.
  - ☑ Test coverage
- Returns the address of the input and error oracle.
  - ☑ Test coverage

Function: `getInputAndError(address token)`

This function is used to get the raw input and error values from the oracle for a given token.

Inputs

- `token`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the token.

Branches and code coverage

Intended branches

- Calls external function `consultLiquidity` from the oracle contract.
  - ☑ Test coverage
- Returns the input and error values from oracle.
  - ☑ Test coverage

Function call analysis

- `consultLiquidity(address token)` (this function is called from the external contract `_inputAndErrorOracle` to get the input and error values)
  - **What is controllable?** `token`.
  - **If the return value is controllable, how is it used and how can it go wrong?** Wrong input and error could mess up the rates and cause the system to behave unexpectedly.
  - **What happens if it reverts, reenters or does other unusual control flow?** This contract does not work properly.

Function: `getSignedInputAndError(address token)`

This function is used to get and transform the input and error values for a given token.

Inputs

- `token`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the token.

Branches and code coverage

Intended branches

- Invokes internal function `getInputAndError` to get the input and error values.

☑ Test coverage

- Invokes the internal function `transformSignedInputAndError` with the token, input, and error values.
  ☑ Test coverage
- Returns the transformed input and error values.
  ☑ Test coverage

Function: `initializePid(address token)`

This function is used to initialize the PID-controller state for a given token.

Inputs

- `token`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the token.

Branches and code coverage

Intended branches

- Invokes the internal function `getSignedInputAndError` to get the signed input and error values.
  ☑ Test coverage
- Sets the last input and error values to the PID state.
  ☑ Test coverage
- Sets the iTerm to the last rate if the buffer size is greater than zero.
  ☑ Test coverage

Function: `needsUpdate(bytes data)`

This function is used to check if the update is needed.

Inputs

- `data`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Data to check if the update is needed — the token address is included in the data.

Branches and code coverage

Intended branches

- Decodes the token address from the data.
  - ☑ Test coverage
- Returns false if the PID config of the token is uninitialized.
  - ☑ Test coverage
- Invokes the parent function `needsUpdate` with the data and returns the result.
  - ☑ Test coverage

Function: `onPaused(address token, bool paused)`

This function is used to reinitialize the PID controller when provided paused parameter is false.

Inputs

- `token`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the token.
- `paused`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Boolean value to check if the token is paused or not.

Branches and code coverage

Intended branches

- Invoke the internal function `initializePid` if the provided paused parameter is false.
  - ☑ Test coverage

Function: `setDefaultInputAndErrorOracle(ILiquidityOracle inputAndErrorOracle_)`

This function is used to set the default input and error oracle address of contract.

Inputs

- `inputAndErrorOracle_`
  - **Control**: Arbitrary.

- **Constraints**: None.
- **Impact**: Address of default input and error oracle address.

Branches and code coverage

Intended branches

- Sets the new default input and error oracle address.
  - ☑ Test coverage

Negative behavior

- Reverts if the caller is not allowed to set the default input and error oracle address.
  - ☑ Negative test
- Reverts if the new default input and error oracle address is zero.
  - ☑ Negative test
- Reverts if the new default input and error oracle address is the same as the old one.
  - ☑ Negative test

Function: `setPidConfig(address token, PidConfig pidConfig)`

This function is used to set the PID configuration for a specific token.

Inputs

- `token`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the token.
- `pidConfig`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Values of the PID-configuration structure.

Branches and code coverage

Intended branches

- Sets the PID configuration for the token.
  - ☑ Test coverage
- Invokes the internal function `initializePid` if the old configuration was uninitialized.
  - ☑ Test coverage

Negative behavior

- Reverts if the caller is not allowed to set the PID configuration.
  - ☑ Negative test
- Reverts if the token's rate buffer's metadata has not been set yet.
  - ☑ Negative test
- Reverts if the PID configuration is invalid.
  - ☑ Negative test

Function: `transformSignedInputAndError(address token, int256 input, int256 err)`

This function is used to transform the input and error values when the transformer is set.

Inputs

- `token`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the token.
- `input`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of input.
- `err`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value or error.

Branches and code coverage

Intended branches

- Calls external function `transformInputAndError` from the transformer contract if the transformer is set.
  - ☑ Test coverage
- Returns the transformed input and error values.
  - ☑ Test coverage

Function call analysis

- `transformInputAndError(int256 input, int256 err)` (this function is called from the external contract `transformer` to transform the input and error values)
  - **What is controllable?** `input` and `err`.

- **If the return value is controllable, how is it used and how can it go wrong?** Wrong transformed input and error could mess up the rates and cause the system to behave unexpectedly.
- **What happens if it reverts, reenters or does other unusual control flow?** This contract does not work properly.

Function: `updateAndCompute(address token)`

This function is used to compute the rate for a given token and update the PID state.

Inputs

- `token`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the token.

Branches and code coverage

Intended branches

- Invokes the internal function `computeNextPidRate` with the token.
  - ☑ Test coverage
- Updates the PID state with the new PID state from the return of `computeNextPidRate`.
  - ☑ Test coverage

Function: `validateInputAndErrorOracle(ILiquidityOracle oracle, bool isDefault)`

This function is used to validate the input and error oracle.

Inputs

- `oracle`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the oracle.
- `isDefault`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Boolean value to check if the oracle is default or not.

Branches and code coverage

Intended branches

- Returns okay if the oracle address is zero when it is default.
  - ☑ Test coverage
- Returns okay if the oracle address is not zero when it is not default.
  - ☑ Test coverage

Negative behavior

- Reverts if the oracle address is zero when it is not default.
  - ☑ Negative test

Function: `willAnythingChange(bytes)`

This function is used to check if anything will change. It is not used in the PID controller, so it always returns true.

Branches and code coverage

Intended branches

- Returns true.
  - ☑ Test coverage

Function: `_inputAndErrorOracle(address token)`

This function is used to get the input and error oracle for a given token.

Inputs

- `token`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the token.

Branches and code coverage

Intended branches

- Returns the default input and error oracle if the token's oracle is not set.
  - ☑ Test coverage
- Returns the token's oracle if the token's oracle is not zero.

☑ Test coverage

Function: `_manuallyPushRate(address token, uint64 target, uint64 current, uint256 amount)`

This function is used to manually push the rates.

Inputs

- `token`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the token.
- `target`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of target member of rate.
- `current`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of current member of rate.
- `amount`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Amount of the rate to be pushed.

Branches and code coverage

Intended branches

- Invokes the parent function `_manuallyPushRate` with the token, target, current, and amount.
  ☑ Test coverage
- Invokes the internal function `initializePid` to reinitialize the PID controller.
  ☑ Test coverage

5.7. Module: PositiveErrorScalingTransformer.sol

Function: `transformInputAndError(int256 input, int256 error)`

This function is used to transform the input and error values as per the scaling factor when the error is positive.

Inputs

- `input`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of input to be transformed.
- `error`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of error to be transformed.

Branches and code coverage

Intended branches

- Scales the error if it is positive.
  - ☑ Test coverage
- Returns the transformed input and error values.
  - ☑ Test coverage

5.8. Module: TrueFiAlocPidController.sol

Function: `push(address alocAddress, RateLibrary.Rate rate)`

This function is used to push a new rate to the controller. It accrues interest for the prior rate before pushing the new rate.

Inputs

- `alocAddress`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Address of the token.
- `rate`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: Value of the rate.

Branches and code coverage

Intended branches

- Calls external function `Interface.automatedLineOfCredit.interestRate` to check if

the alocAddress token contract has an interface of interest rate.

☑ Test coverage

- Calls external function `Interface.automatedLineOfCredit.updateAndPayFee` to update and pay the fee in alocAddress token contract if the alocAddress token contract has an interface of interest rate.

  ☑ Test coverage

- Invokes the parent `push` function.

  ☑ Test coverage

Function call analysis

- `interestRate()` (this function is called to check if the alocAddress token contract has an interface of interest rate)
  - **If the return value is controllable, how is it used and how can it go wrong?** Update and pay-fee process for token does not work properly.
  - **What happens if it reverts, reenters or does other unusual control flow?** This contract does not work properly.
- `updateAndPayFee()` (this function is called to update and pay the fee in alocAddress token contract)
  - **What happens if it reverts, reenters or does other unusual control flow?** This contract does not work properly.

6. Assessment Results

At the time of our assessment, the reviewed code was not deployed to the Ethereum Mainnet.

During our assessment on the scoped Adrastia PID Controller contracts, we discovered two findings, both of which were low impact.

6.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.