yAudit Timeless Finance Gauge Review

Review Resources:

Codebase from three different repositories.

Auditors:

- pandadefi
- prady

Table of Contents

Review Summary

Timeless Finance

Timeless gauges are emulating the mechanics of curve multi-chain gauges from Curve, with a handful of modifications. Small adjustments have been made to the gauge contracts to facilitate updates to the boost, as well as eliminating the reliance on anyCall. Additionally, they've independently created their own ve balance cross-chain contracts.

The contracts of three timeless repositories were reviewed over 21 days. The code review was performed by 2 auditors between May 22 and June 9, 2023. The repository was under active development during the review, but the review was limited to the latest commit at the start of the review. This was commit 5236bc3d6bcd331d1ebf1bc8e80fe61163f937c2 for the ve-beacon repo, commit fb5f03691972ad3d3edce534f236a47c586eecaf for the universal-bridge-lib repo, and commit 41344e3197ce7b007f390c15360234a6e159ad65 for the gauge-foundry repo.

Scope

The scope of the review consisted of the following contracts in three different repositories:

- ve-beacon - commit 5236bc3d6bcd331d1ebf1bc8e80fe61163f937c2

```
├── src
│   ├── VeBeacon.sol
│   ├── VeRecipient.sol
│   ├── base
│   │   ├── BoringOwnable.sol
│   │   └── Structs.sol
```

```
│   ├── interfaces
│   │   └── IVotingEscrow.sol
│   └── recipients
│       ├── VeRecipientAMB.sol
│       ├── VeRecipientArbitrum.sol
│       ├── VeRecipientOptimism.sol
│       └── VeRecipientPolygon.sol
```

- universal-bridge-lib - fb5f03691972ad3d3edce534f236a47c586eecaf

```
├── src
│   ├── UniversalBridgeLib.sol
│   └── bridges
│       ├── ArbitraryMessageBridge.sol
│       ├── ArbitrumBridge.sol
│       ├── OptimismBridge.sol
│       └── PolygonBridge.sol
```

- gauge-foundry - 41344e3197ce7b007f390c15360234a6e159ad65

```
├── vyper_contracts
│   ├── ChildGauge.vy
│   ├── ChildGaugeFactory.vy
│   ├── GaugeController.vy
│   ├── RootGauge.vy
│   ├── RootGaugeFactory.vy
│   └── bridgers
│       ├── ArbitrumBridger.vy
│       ├── ArbitrumRefund.vy
│       ├── OmniBridger.vy
│       ├── OptimismBridger.vy
│       └── PolygonBridger.vy
```

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged

accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, Timeless Finance and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	Properly access controlled. One case of DoS should be fixed.
Mathematics	Good	The mathematical operations are secure with Solidity 0.8 and Vyper. The Ve Beacon contracts adopt the decay mechanisms found in VotingEscrow.vy.
Complexity	Good	Contracts are well designed.
Libraries	Average	The utilization of libraries has been executed effectively.
Decentralization	Good	The protocol can function without the need for the admin.
Code stability	Good	All key features have been successfully implemented, and the code has demonstrated stability throughout the review process.
Documentation	Average	We didn't have access to documentation during the audit.
Monitoring	Average	Some functions are missing event emission.
Testing and verification	Low	Conducting tests on fund flows using suitable bridges is recommended, even though we acknowledge the

Category	Mark	Description
		complexity and difficulty involved in this task.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
  - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements
- Gas savings
  - Findings that can improve the gas efficiency of the contracts
- Informational
  - Findings including recommendations and best practices

Critical Findings

None.

High Findings

1. High - VeBeacon contract vulnerable to denial of service (DoS) attack

Using self-destruct, an attacker can send ETH to the contract, making it unusable.

Technical Details

The following code reverts when ETH is left on the contract.

```
        if (address(this).balance != 0) revert VeBeacon__LeftoverEth();
```

This ensures users don't overpay for the Arbitrum bridge, but it has two issues:

- An attacker will be able to DOS the contract by sending some ETH dust that will remain, causing the check to revert.
- A user might get his transaction reverted because the Arbitrum fee might change between the time the transaction is broadcasted and the time it gets executed.

VeBeacon.sol#L66

Impact

High. The contract can become unusable.

Recommendation

Refund the unused ETH if the cost of the gas to send back the ETH is lower than the amount to be sent back. This can be done using the `block.basefee` and the standard gas cost of a transfer of 21k.

Developer Response

Fixed in https://github.com/timeless-fi/ve-beacon/pull/1/commits/a47a2a6a1df43dc45fb524e5e9692687a5feb81f

Medium Findings

1. Medium - Potential denial of service vulnerability in bridger contracts

The Bridger contracts use the `ERC20(_token).transferFrom()` and `ERC20(_token).approve()` function calls. Some tokens like USDT do not return a value for these calls, resulting in the assert statement constantly failing. This could cause a Denial of Service (DoS) for these tokens.

Technical Details

When a user tries to bridge tokens like USDT that doesn't return a boolean value on `transferFrom()` and `approve()` function calls, the operation will fail since the Bridger contracts use the assert statement to validate the function call's success. In Vyper, the assert function throws an exception if the condition is unmet and reverts all changes.

Impact

Medium. Potential incompatibility or denial of service for certain tokens.

Recommendation

1 A recommended fix is to make a low-level raw_call to the ERC20 token contracts. This will help manage the return value properly, as it gives you more control over the execution of the contract call. This approach is particularly useful when working with contracts that might not fully comply with the standard, like USDT. Here's an example of how it can be implemented in Vyper:

```
response: Bytes[32] = raw_call(
    _token,
    _abi_encode(
        msg.sender, self, _amount,
method_id=method_id("transferFrom(address,address,uint256)")
    ),
    max_outsize=32,
)
if len(response) != 0:
    assert convert(response, bool)
```

This is just an example to show the approach. You need to adapt this example to fit in the actual contract code.

1 Another solution could be to use the default_return_value option in the transferFrom and approve function calls. Here's how it could be implemented in Vyper:

```
ERC20(_token).transferFrom(msg.sender, self, _amount, default_return_value=True)
```

However, be cautious when using the default_return_value option, as it could potentially mask other issues. If the transferFrom function does not return a value due to a problem unrelated to tokens like USDT, this code will still consider the function call successful. Use this option carefully and only if it is suitable for your specific context.

Developer Response

Fixed in https://github.com/timeless-fi/gauge-foundry/pull/1/commits/b982da598c2cf8da51e6c473f701a84400b29895

2. Medium - Child gauge vulnerability: uncontrolled reward addition by managers

With the current design, any user can create a child gauge and become the manager of

that child gauge. While the team can later change the manager using `set_manager(_manager: address)`, before the update occurs, a manager can add multiple rewards by calling `add_reward(_reward_token: address, _distributor: address)` and reach the limit of rewards defined as `MAX_REWARDS`. This can prevent the distribution of actual rewards.

Technical Details

Refer to ChildGauge.vy#L579 for the relevant code snippet.

Impact

Medium. Lack of access control for gauge creation and initial ownership can be problematic.

Recommendation

Consider one of the following options:

- Implement a remove reward function.
- Restrict access to `deploy_gauge` to trusted entities.
- Set the default manager as the factory owner.

Developer Response

Fixed in https://github.com/timeless-fi/gauge-foundry/pull/1/commits/a998eaa6372287f7224bdc54302851900aa70d1d

3. Medium - unkillGauge should checkpoint funds distribution to prevent distributing for the time it was killed

When a gauge is restored using the unkillGauge() method, users of that gauge can claim rewards retroactively, back to the point when the gauge was 'killed'.

Technical Details

The function unkillGauge() merely changes the gauge state flag to active. User reward claims will trigger a checkpoint that will distribute rewards for the period up until the last checkpoint, which was set when the gauge was 'killed'. Consequently, this distributes rewards corresponding to the duration in which the pool was inactive.

Impact

Medium. Funds could be distributed for the entire duration the pool was in the 'killed' state.

Recommendation

We suggest updating the `period` and `period_timestamp` state variables when the `unkillGauge` method is invoked.

Developer Response

Fixed in https://github.com/timeless-fi/gauge-foundry/pull/1/commits/f2fe0a1b49b01824eff5894746976202648c1aff

4. Medium - Potential user exploitation of boost adjustment mechanism

In the ChildGauge module, the team implemented a method called set_tokenless_production(). This function alters the boost value for a gauge. When the boost is decreased, any user not actively interacting with the gauge will continue to benefit from the previous boost level until their next interaction with the gauge.

Technical Details

The function at ChildGauge.vy#L699 sets a new boost value. While a 'kick' function exists, it cannot be utilized to update a user's effective balance. The kick() method can only be used when the user has updated a lock.

Impact

Medium. Users can exploit these mechanics to earn more rewards.

Recommendation

We recommend modifying the 'kick' function to allow for the update of the effective balance if it has diverged from the original effective balance by a predetermined percentage.

Developer Response

Fixed in https://github.com/timeless-fi/gauge-foundry/pull/1/commits/a03f79ba84895fcfc4fc295e7f3cbb48386fce85

Low Findings

1. Low - Rewards can be locked in Gauges

There's a risk that rewards may become inaccessible when depositing smaller quantities into the gauge.

Technical Details

`childGuage.vy` function deposit_reward_token allows anyone to create, and rewards distribution is done using the reward rate. Still, if the reward `distributor` deposits `_amount` (in WEI) less than `1 WEEK` (or 604800 Wei), the transferred rewards will be locked into the contract.

```
self.reward_data[_reward_token].rate = _amount / WEEK
```

If `_amount < 604800` then `self.reward_data[_reward_token].rate = 0`.

Impact

Low. This will be fine for tokens with higher decimals but can become significant for smaller ones likeUSDC [6 decimals], GUSD [2 decimals].

Recommendation

Use scaling factor for reward rate with scale >= 1e6.

```
self.reward_data[_reward_token].rate = (_amount * REWARD_SCALE) / WEEK
```

Developer Response

Fixed in https://github.com/timeless-fi/gauge-foundry/pull/1/commits/88dcc705686bf0c6ff134821428250fbe6bd789b

2. Low - Add missing input validation on constructor/initializer/setters

Validating inputs is critical prior to establishing important variables.

Technical Details

Certain functions are missing necessary input validation.

- In childGaugeFactory.vy: `set_implementation`, `set_voting_escrow`, `deploy_gauge`, and `__init__`.
- In childGuage.vy: `__init__`, `add_reward`, `set_manager`, `initialize`.

Impact

Low. Proper input validation can help revert when important state variables are about to be set with wrong values.

Recommendation

Consider implementing all the checks suggested above.

Developer Response

Acknowledged, won't fix.

3. Low - Potential user manipulation of Uniswap deposit to claim rewards within range

A Gauge can be configured for automated distribution of rewards, which triggers when a Uniswap v3 deposit is within the swap range. However, a user could manipulate this system to claim rewards by sandwiching an out-of-range deposit, thereby garnering rewards from the gauge.

Technical Details

For more information, refer to ChildGauge.vy#L744.

Impact

Low. Funds could be distributed inappropriately.

Recommendation

As we don't have access to `UNISWAP_POOR_ORACLE`, it could be used as a safeguard against price manipulation within the same block or transaction.

Developer Response

The Uniswap Price-out-of-range oracle is secure against sandwich attacks, so such manipulation is not feasible. See https://github.com/timeless-fi/uniswap-poor-oracle

Gas Findings

1. Gas - votingEscrow.epoch() can't be zero if a user created a lock

`_broadcastVeBalance()` function currently checks for the votingEscrow global epoch after verifying the user epoch. However, since creating a lock always increases the global epoch, the assert statement is unnecessary.

Technical Details

The following code snippet is not necessary:

```
if (epoch == 0) revert VeBeacon__EpochIsZero();
```

This check is redundant as a validation already exists in line 117. If a user has created a lock, the global epoch cannot be zero, as confirmed in the ve contract.

VeBeacon.sol#L122

Impact

Gas savings.

Recommendation

Consider removing the assert statement as it is unnecessary.

Developer response

Fixed in https://github.com/timeless-fi/ve-beacon/pull/1/commits/6395e5a5f7c973bc99bc98cdd4005d3cec6bb1e3

2. Gas - Consolidating Gauge State Setters for Gas Optimization

Multiple functions are made to set the same value, they could be grouped together.

Technical Details

Multiple setters are currently used to modify the `gauge_state` value. To optimize gas usage, it is recommended to consolidate these setters into a single function.

Impact

Gas savings.

Recommendation

To reduce code size and enhance efficiency, consider utilizing a Vyper enum instead of numbers and implement a single setter function.

Developer response

Acknowledged, won't fix.

Informational Findings

1. Informational - Potential transaction order discrepancy and impact on total

supply calculation in sidechain

In an extreme case where the transaction order is not respected on the sidechain, the total supply might not be correct for a short period of time.

Technical Details

There is no warranty on transaction ordering from L1 to L2. A transaction can be mined on L1, fail on L2, and then be retried later, resulting in a different order.

Here is an example:

- User A creates a lock that will last for 4 years.
- User A broadcasts their lock to L2.
- User B creates a lock that will last two weeks.
- User B broadcasts their lock to L2.

If the broadcast from User B is processed by L2 before the one from User A, then the slope changes corresponding to the end of the lock from User B will be removed from the corresponding variable when User A's transaction is processed on L2.

Impact

Informational.

Recommendation

Monitor this potential edge case. The impact of such a small lock is negligible, as the user's balance will go to zero regardless. The total supply will be fixed once a new update is sent to L2.

Developer response

Acknowledged.

2. Informational - Optimizing bytecode size by removing unused constant variable

Having an unused constant variable not only increases the size of the bytecode but can also confuse developers with unnecessary variables.

Technical Details

There is an unused constant variable in the contract childGaugeFactory.vy that should be removed.

Impact

Informational. Removing the unused variable will result in a reduction in the deployed bytecode size.

Recommendation

Remove the unused variable from the contract.

Developer response

Fixed in https://github.com/timeless-fi/gauge-foundry/pull/1/commits/76baf36dbddb1b496cb2e9e0034c6b2ee59a1730

3. Informational - Missing events for critical operations

Several critical operations do not trigger events. As a result, it is difficult to check the behavior of the contracts.

Technical Details

Ideally, the following critical operations should trigger events.

In `VeRecipient`

- `constructor()` should emit `SetBeacon`

In `childGuage`, following function lacks event emission.

- `killGauge()`
- `unkillGauge()`
- `makeGaugePermissionless()`
- `set_manager()`
- `deposit_reward_token()`
- `kick()`
- `set_reward_distributor()`
- `add_reward()`
- `claim_rewards()`
- `set_rewards_receiver()`

In `rootGuage`, following functions lacks event emission.

- `set_killed()`

Impact

Informational. Without events, users and blockchain-monitoring systems cannot easily detect suspicious behavior.

Recommendation

Add events for all critical operations. Events aid in contract monitoring and the detection of suspicious behavior.

Developer response

Fixed in https://github.com/timeless-fi/gauge-foundry/pull/1/commits/25225cfce5af64cfce655dd63089771d90026d20 and https://github.com/timeless-fi/ve-beacon/pull/1/commits/6b558e15076589eec70f1612d545253931758e4a

4. Informational - Leveraging new syntax in Vyper 0.3.4 for default return value

The latest version, vyper 0.3.4, introduces a new syntax that allows the addition of a default_return_value to a call.

Technical Details

The existing codebase uses low-level raw_calls for safe calls. This can be simplified using the new syntax.

Impact

Informational.

Recommendation

Consider employing the new syntax:

```
response: Bytes[32] = raw_call(
                        token,
                        _abi_encode(
                            receiver,
                            total_claimable,
                            method_id=method_id("transfer(address,uint256)")
                        ),
                        max_outsize=32,
```

```
                    )
                    if len(response) != 0:
                        assert convert(response, bool)
```

This can be replaced by:

```
assert ERC20(token).transfer(receiver, total_claimable, default_return_value=True)
```

Developer response

Fixed in https://github.com/timeless-fi/gauge-foundry/pull/1/commits/96780f1c28043f8a9f55595f02ef5019f807ae1b

Final remarks

In summary, the code review of the Timeless Finance contracts revealed findings of varying impact levels. While the contracts exhibit strong design and implementation, featuring proper access control, secure mathematical operations, and decentralization capabilities, there are specific areas that need to be addressed to enhance functionality and security.