PUBLIC

Code Assessment

of the yBAL Smart Contracts

JUNE 13, 2023

Produced for

by

Contents

1 Executive Summary 3
2 Assessment Overview 5
3 Limitations and use of report 8
4 Terminology 9
5 Findings 10
6 Resolved Findings 12
7 Open Questions 14
8 Informational 15
9 Notes 16

1 Executive Summary

Dear Yearn team,

Thank you for trusting us to help Yearn with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of yBAL according to Scope to support you in forming an opinion on their security risks.

Yearn implements a liquid wrapper for Balancer's voting tokens (veBAL) and an ecosystem allowing users to invest the funds to earn a yield on their deposits. A Zapper contract allows a convenient entry point for users to invest their funds. The new liquid wrapper token yBAL can be always minted by providing BAL or WETH tokens. Having yBAL the users can decide between the associated investment strategies and earn from LP rewards or staking rewards.

No critical issues were uncovered in the intermediate audit. In summary, we find that the current intermediate codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	5
• Code Corrected	2
• Code Partially Corrected	2
• Risk Accepted	1

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the yBAL repository based on the documentation files.

The scope consists of five smart contracts in the `contract` directory:

1. `BalancerYBALVoter.sol`
2. `yBAL.vy`
3. `Zap.vy`
4. `StrategyProxy.sol`
5. `StrategyStYBAL.sol`

The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	May 24th 2023	3a84a89c5e3e3cf250cbd0e45cd938928249b980	Initial Version
2	June 12th 2023	55557c18d9c941c572c709d7eb28acc905d99f06	Version 2

For the solidity smart contracts, the compiler version `^0.8.15` was chosen. For the vyper smart contracts, the compiler version `0.3.7` was chosen.

2.1.1 Excluded from scope

Any other file not explicitly mentioned in the scope section. In particular tests, scripts, external dependencies, and configuration files are not part of the audit scope.

2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Balancer's `BAL-WETH` liquidity tokens can be locked for up to 1 year in a `VotingEscrow` contract. Such locking mints `veBAL` tokens. These non-transferable tokens receive a share of trading fees from the Balancer protocol. In addition, `veBAL` tokens can boost rewards on liquidity provided to the Balancer protocol when token holders participate in governance, which allows them to direct the `BAL` rewards toward selected pools. However, once locked the liquidity tokens cannot be transferred or withdrawn from the escrow.

Yearn offers a system to make these voting shares more liquid by allowing users to wrap their `BAL-WETH` token into `yBAL` which can then be staked into strategies to make the most out of the Balancer rewards

while remaining able to swap the wrapped token at any time on the market. Additionally, a zapper to swap between the different Yearn tokens of the Balancer ecosystem is provided.

2.2.1 yBAL Token Wrapper

The yBAL token is an ERC20 wrapper for the `BAL-WETH` Balancer liquidity token, it is an intermediary token to allow liquid positions on Balancer's governance and rewards. At any point in time, users can mint 1 `yBAL` in exchange for 1 `BAL-WETH` through the `mint()` function. Note that no burning mechanism exists to redeem.

2.2.2 yBAL Voter

The voter contract will receive all `BAL-WETH` tokens that were exchanged to mint `yBAL` tokens and lock them in the Balancer escrow to receive rewards. It allows the governance or some whitelisted strategies to vote on certain gauges or to make arbitrary state-modifying calls from the voter. It is a critical part of the ecosystem as it is the foundation on which the (theoretical) peg of `yBAL` is based. A permissioned function named `execute()` allows a caller to execute arbitrary calls from the voter contract.

2.2.3 Strategy Proxy

The StrategyProxy smart contract is used as an intermediary contract to manage the voter's interaction and its voting power usage. For this, multiple permission variables exist from which the most important ones are:

1. Lockers: A mapping that allows an address to lock the existing `BAL-WETH` balance in the voter, and to increase the locking time to the maximum.
2. Voters: A mapping that allows an address to vote for any gauge with any weight with the voter's voting power.
3. Strategies: A mapping that allows an address to manage the voter's position on a specific Balancer gauge, making it possible to deposit and withdraw the underlying token, as well as claim rewards from the gauge.
4. Fee Recipient: The fee recipient can decide where to send the trading rewards and when to claim them.
5. Factory: Balancer vault factory, allowing to approve of any strategy for a specific gauge.
6. Governance: Manages all permissions.

All external functions are permissioned and most critical actions are executed through the voter's `execute()` function.

2.2.4 Strategy Staked yBAL

Users with a yBAL balance do not profit in any way from rewards or voting power of the Balancer ecosystem because the token is only an ERC20 wrapper and does not have underlying logic. Consequently, a Yearn vault exists for this token, the `StYBAL` token. This vault's strategies can then manage the underlying `BAL-WETH` as needed (through the StrategyProxy) to make the most out of the Balancer rewards and voting power.

StrategyStYBAL smart contract is a strategy designed for the `StYBAL` token. It receives the trading fees and rewards from the voter's lock position and compounds them back to yBAL. Note that to be able to achieve this it needs to have the `feeRecipient` role in the StrategyProxy.

The compounding logic is outsourced to Yearn's trading factory.

2.2.5 Zapper

Yearn's Balancer ecosystem contains multiple tokens that wrap different strategies or positions. The zapper smart contract is a converter that allows to swap between each of them. Users can specify an `input_token` and an `output_token` as well as a minimum output amount and call the `zap()` function on the contract without having to care about internal details. Note that input tokens and output tokens are verified to be in a token whitelist, to make sure there is an existing path to swap between them.

2.2.6 Trust Model

The governance is assumed to be a fully trusted multi-sig or similar setup. It has every right over all `BAL-WETH` locked in the Balancer escrow through the voter and the StrategyProxy. It can also execute any arbitrary call from the voter contract, as well as add or revoke any permissions from the proxy. Most Yearn vaults also allow the governance role to set parameters freely. Therefore, the Yearn governance is trusted not to act maliciously.

All permissioned addresses in the StrategyProxy could potentially hurt the ecosystem if managed poorly or if acting maliciously, for this the governance is also trusted to only add some trusted and functionally correct actors.

Note that no permissioned actor can temper with the yBAL token or the zapper contracts in any way once deployed. However, a sweeping functionality is provided.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	3

- Missing Sanity Checks Code Partially Corrected
- No Event Emitted for State Modifying Code Code Partially Corrected
- Vault Withdrawal Could Fail in the Zapper Risk Accepted

5.1 Missing Sanity Checks

Correctness Low Version 1 Code Partially Corrected

Multiple setter functions are missing sanity checks. The setters functions are permissioned and we assume the caller to be trusted. Still, mistakes can happen and would be irreversible in the following cases:

- `StrategyProxy.setGovernance`
- `BalancerYBALVoter.setGovernance`

In other cases it might be helpful to prevent setting an address accidentally to `address(0)` but it is easy to override the value because the role setting the new address is not changed and could reverse their mistake. E.g., in `StrategyProxy.setFeeToken` it is possible to accidentally set it to `address(0)` but it could be immediately corrected by governance. Still, this might cause side effects as transactions could be executed with incorrectly set values before the mistake is reversed.

In `BalancerYBALVoter`, neither the initializing function nor the setters do verify that their address inputs are different than the zero address. Using sanity checks prevents unfortunate errors that could potentially brick the contract.

In the zapper contract the `_recipient` is not checked and might be `address(0)` through a UI problem or incorrect user call.

Code partially corrected

A sanity check was added in the `zap` function to prevent the `_recipient` from being `address(0)`. All other sanity checks were acknowledged by client but remained unchanged.

5.2 No Event Emitted for State Modifying Code

Design | Low | Version 1 | Code Partially Corrected

Events indicate major state changes. Hence, it might be useful for users to listen to certain events. But events do increase the gas costs slightly. Yearn might consider the option to add events in the following cases:

- `BalancerYBALVoter` smart contract allows for modification of the `governance` and the `strategy` variables, but does not emit any event along with such modifications.
- Some of the `StrategyStYBAL` setters like `setProxy` might issue events

Code partially corrected

An event was added that indicated a change of the `governance` variable in `BalancerYBALVoter`. In all other cases Yearn decided to keep the code as it is.

5.3 Vault Withdrawal Could Fail in the Zapper

Design | Low | Version 1 | Risk Accepted

Vaults such as `StYBAL` mint some shares when users deposit some underlying into the contract. To withdraw underlying tokens from such vaults, user specify an amount of shares and the smart contract computes the underlying value by using an exchange rate that depends on total funds available and profit/loss.

Note that in the zapper, there exists the following assertion after withdrawing `_amount_in` shares from a vault:

```
assert amount >= _amount_in # dev: fail on partial withdrawal
```

This code snipped could make a zapping transaction revert for two reasons:

1. A small rounding down can happen when computing the underlying value, which would make `amount` a little bit smaller than `_amount_in`.
2. Some loss could have occured in the strategies, which could make 1 share of the vault worth less than 1 underlying.

Risk accepted

Yearn is aware of the issue and accepts the risk as they rate the probability of the issue very low.

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	2

- Missing Parameter in JoinPool Struct Code Corrected
- Outdated Comments Left Code Corrected

6.1 Missing Parameter in JoinPool Struct

Correctness Low Version 1 Code Corrected

The `userData` field in Balancer's JoinPool struct depends on the type of join, in the case of an `EXACT_TOKENS_IN_FOR_BPT_OUT` join type, the field must be organised encoded in this way: `[EXACT_TOKENS_IN_FOR_BPT_OUT, amountsIn, minimumBPT]`.

In the zapper's `_lp_balweth()` function, only the join type and the amounts are encoded into the `userData`:

```
user_data: Bytes[160] = _abi_encode(
     convert(1, uint8), # EXACT_TOKENS_IN_FOR_BPT_OUT
     _amounts,
 )
```

Meaning that in the final encoding form, the length of the amounts will be accounted for as `minimumBPT`.

Code corrected

User data encoding was changed and `convert(0, uint256)` was added as `minimumBPT` amount.

6.2 Outdated Comments Left

Correctness Low Version 1 Code Corrected

Some curve-related comments are left in the code:

1. StrategyStYBAL line 171
2. StrategyProxy line 60, 96

Code corrected

Code comments were removed or changed accordingly.

6.3 Missing Capital Letter

Informational Version 1 Code Corrected

The `setdisableClaim()` function in the `StrategyStYBAL` contract does not respect the camel casing. It should be `setDisableClaim()`.

Code corrected

The function was renamed correctly.

7 Open Questions

Here, we list open questions that came up during the assessment and that we would like to clarify to ensure that no important information is missing.

7.1 Inconsistent Call Behavior

Open Question **Version 1**

The `StrategyProxy` contract calls the voter (confusingly referred as `proxy` in the code) to execute various operations. The voter contract has a function to execute arbitrary calls. In most execution this is used. E.g., `vote_for_gauge_weights`, `withdraw`, `transfer`, `approve`, `deposit` and `mint`. But for `increaseAmount` a special function is defined in the voter contract. Why is this design chosen?

7.2 State Inconsistency for Safe Tokens

Open Question **Version 1**

The functions `approveRewardToken` and `approveExtraTokenRecipient` check if a token is safe by calling `_isSafeToken`. This guarantees that the tokens are considered safe at this moment in time (`t0`) when calling the functions. The check relies on the state of two other contracts to check if a token is safe. When operations with the tokens are performed later in time (`t0 + x`), the check is not repeated.

Hence, it implicitly assumes that the state of the two contracts does not change after a token was checked in `_isSafeToken`. Please provide a brief explanation why this assumption will always hold.

8 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

8.1 No Error Strings Returned in the Zapper

Informational Version 1

Meaningful error messages help users to understand why a traansaction failed. No error strings are returned for assertions in the Zapper contract. Some might be helpful for the users to be able to understand the reason for the failed transaction, especially in the case of slippage protection.

8.2 Variables Could Be Immutable

Informational Version 1 Code Partially Corrected

Setting variables that are only set in the constructor and cannot be changed later to immutable will save gas as the value is hardcoded into the contract and no SLOAD is required. Such variables could be:

- The `name` and the `sweep_recipient` variables of the Zap contract
- The variables `escrow`, `token` and `name` of the Voter contract
- The `name`, `symbol` and `decimals` variables of the yBAL smart contract

---

Core partially corrected

The sweep recipient variable was made immutable. Yearn left all other variables as they are because the contracts are already deployed.

9 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

9.1 Balancer LP Tokens Cannot Be Claimed as Extra Token Rewards

Note Version 1

The `StrategyProxy` 's `_isSafeToken()` function makes sure a token isn't a gauge or a Balancer liquidity token. Because of that, no Balancer liquidity token can be claimed as extra rewards in the `StrategyProxy`. However, such extra rewards could potentially exist to claim (via a bribe for example), and would be lost in this case.