MixBytes()

Yield Basis DAO Security Audit Report

AUGUST 11, 2025

Table of Contents

1. Introduction

1.1 Disclaimer

The audit makes no statements or warranties regarding the utility, safety, or security of the code, the suitability of the business model, investment advice, endorsement of the platform or its products, the regulatory regime for the business model, or any other claims about the fitness of the contracts for a particular purpose or their bug-free status.

1.2 Executive Summary

Yield Basis DAO lets users lock YB token for voting power and direct emissions to pools via gauge voting. Therefore, in this audit, we paid special attention to attacks related to voting power manipulation, gauge weight manipulation, and reward distribution mechanisms. We also went through our detailed checklist, covering other aspects such as business logic, common ERC20 issues, interactions with external contracts, integer overflows, reentrancy attacks, access control, typecast pitfalls, rounding errors and other potential issues.

Key notes and recommendations:

- We suggest adding additional boundary tests, preferably without mocking the main contracts.
- The `LiquidityGauge` requires a minimum of `10**12` shares for the initial mint; however, this is acceptable given that the underlying asset is an 18-decimal LP token, making 1e12 wei a relatively small value.

1.3 Project Overview

Summary

Title	Description
Client Name	Yield Basis
Project Name	DAO
Type	Vyper
Platform	EVM
Timeline	24.06.2025 - 01.08.2025

Scope of Audit

File	Link
contracts/dao/GaugeController.vy	GaugeController.vy
contracts/dao/LiquidityGauge.vy	LiquidityGauge.vy
contracts/dao/YB.vy	YB.vy
contracts/dao/VotingEscrow.vy	VotingEscrow.vy
contracts/dao/CliffEscrow.vy	CliffEscrow.vy
contracts/dao/VestingEscrow.vy	VestingEscrow.vy
contracts/Factory.vy	Factory.vy
contracts/dao/erc4626.vy	erc4626.vy

Versions Log

Date	Commit Hash	Note
24.06.2025	3352c612fc33e48f1a106da41f63810f31bc38be	Initial Commit
22.07.2025	4378752a0bb17169648a8711598a18372a93de7f	Commit for re-audit
01.08.2025	a97447e0d075cd584c2370ee8ac0c5bf1b8c8c19	Commit for re-audit 2

Mainnet Deployments

The deployment verification will be conducted later after the full deployment of the protocol into the mainnet.

1.4 Security Assessment Methodology

Project Flow

Stage	Scope of Work
Interim audit	**Project Architecture Review:** · Review project documentation · Conduct a general code review · Perform reverse engineering to analyze the project's architecture based solely on the source code · Develop an independent perspective on the project's architecture · Identify any logical flaws in the design OBJECTIVE: UNDERSTAND THE OVERALL STRUCTURE OF THE PROJECT AND IDENTIFY POTENTIAL SECURITY RISKS.
	Code Review with a Hacker Mindset: · Each team member independently conducts a manual code review, focusing on identifying unique vulnerabilities. · Perform collaborative audits (pair auditing) of the most complex code sections, supervised by the Team Lead. · Develop Proof-of-Concepts (PoCs) and conduct fuzzing tests using tools like Foundry, Hardhat, and BOA to uncover intricate logical flaws. · Review test cases and in-code comments to identify potential weaknesses. OBJECTIVE: IDENTIFY AND ELIMINATE THE MAJORITY OF VULNERABILITIES, INCLUDING THOSE UNIQUE TO THE INDUSTRY.
	Code Review with a Nerd Mindset: · Conduct a manual code review using an internally maintained checklist, regularly updated with insights from past hacks, research, and client audits. · Utilize static analysis tools (e.g., Slither, Mythril) and vulnerability databases (e.g., Solodit) to uncover potential undetected attack vectors. OBJECTIVE: ENSURE COMPREHENSIVE COVERAGE OF ALL KNOWN ATTACK VECTORS DURING THE REVIEW PROCESS.

Stage	Scope of Work
	Consolidation of Auditors' Reports: · Cross-check findings among auditors · Discuss identified issues · Issue an interim audit report for client review OBJECTIVE: COMBINE INTERIM REPORTS FROM ALL AUDITORS INTO A SINGLE COMPREHENSIVE DOCUMENT.
Re-audit	**Bug Fixing & Re-Audit:** · The client addresses the identified issues and provides feedback · Auditors verify the fixes and update their statuses with supporting evidence · A re-audit report is generated and shared with the client OBJECTIVE: VALIDATE THE FIXES AND REASSESS THE CODE TO ENSURE ALL VULNERABILITIES ARE RESOLVED AND NO NEW VULNERABILITIES ARE ADDED.
Final audit	**Final Code Verification & Public Audit Report:** · Verify the final code version against recommendations and their statuses · Check deployed contracts for correct initialization parameters · Confirm that the deployed code matches the audited version · Issue a public audit report, published on our official GitHub repository · Announce the successful audit on our official X account OBJECTIVE: PERFORM A FINAL REVIEW AND ISSUE A PUBLIC REPORT DOCUMENTING THE AUDIT.

1.5 Risk Classification

Severity Level Matrix

Severity	Impact: High	Impact: Medium	Impact: Low
Likehood: High	Critical	High	Medium
Likehood: Medium	High	Medium	Low
Likehood: Low	Medium	Low	Low

Impact

- **High** – Theft from 0.5% OR partial/full blocking of funds (>0.5%) on the contract without the possibility of withdrawal OR loss of user funds (>1%) who interacted with the protocol.
- **Medium** – Contract lock that can only be fixed through a contract upgrade OR one-time theft of rewards or an amount up to 0.5% of the protocol's TVL OR funds lock with the possibility of withdrawal by an admin.
- **Low** – One-time contract lock that can be fixed by the administrator without a contract upgrade.

Likelihood

- **High** – The event has a 50–60% probability of occurring within a year and can be triggered by any actor (e.g., due to a likely market condition that the actor cannot influence).
- **Medium** – An unlikely event (10–20% probability of occurring) that can be triggered by a trusted actor.
- **Low** – A highly unlikely event that can only be triggered by the owner.

Action Required

- **Critical** – Must be fixed as soon as possible.
- **High** – Strongly advised to be fixed to minimize potential risks.
- **Medium** – Recommended to be fixed to enhance security and stability.
- **Low** – Recommended to be fixed to improve overall robustness and effectiveness.

Finding Status

- **Fixed** – The recommended fixes have been implemented in the project code and no longer impact its security.
- **Partially Fixed** – The recommended fixes have been partially implemented, reducing the impact of the finding, but it has not been fully resolved.
- **Acknowledged** – The recommended fixes have not yet been implemented, and the finding remains unresolved or does not require code changes.

1.6 Summary of Findings

Findings Count

Severity	Count
Critical	0
High	4
Medium	2
Low	5

Findings Statuses

ID	Finding	Severity	Status
H-1	Uninitialized `specific_emissions_per_gauge` in `GaugeController.add_gauge()`	High	Fixed
H-2	Loss of Rewards When `LiquidityGauge.totalSupply=0`	High	Fixed
H-3	Infinitely Locked `VotingEscrow` Positions Are Not Accounted During Voting in `GaugeController`	High	Fixed
H-4	Flash Loan Attack on Gauge Reward Distribution via `get_adjustment()` Manipulation	High	Fixed
M-1	Inflation Attack on LiquidityGauge	Medium	Fixed
M-2	Incorrect Merging of Lock Ends	Medium	Fixed
L-1	Documentation and Code Inconsistencies	Low	Fixed
L-2	Missing Zero-Address Checks in `Factory` Constructor	Low	Fixed
L-3	`unlock_time` Can Exceed 4-Year Cap in `CliffEscrow`	Low	Fixed
L-4	One-Step Ownership Transfer on Critical Contracts	Low	Acknowledged

L-5	Incorrect Bound Checking in Factory.add_market()	Low	Fixed

2. Findings Report

2.1 Critical

Not Found

2.2 High

H-1	Uninitialized specific_emissions_per_gauge in GaugeController.add_gauge()		
Severity	High	**Status**	Fixed in 4378752a

Description
New gauges don't initialize specific_emissions_per_gauge[gauge], allowing attackers to backrun add_gauge(), vote, and claim excess rewards in the same block, due to the extra difference between specific_emissions - self.specific_emissions_per_gauge[gauge] (which is specific_emissions - 0):

```
if block.timestamp > t:

    self.weighted_emissions_per_gauge[gauge] +=
      (specific_emissions - self.specific_emissions_per_gauge[gauge])
      * aw // 10**18

    self.specific_emissions_per_gauge[gauge] = specific_emissions
```

GaugeController.vy#L187
Recommendation
We recommend initializing specific_emissions_per_gauge[gauge] in add_gauge().

H-2	Loss of Rewards When LiquidityGauge.totalSupply=0		
Severity	High	**Status**	Fixed in 4378752a

Description

The vulnerability lies in LiquidityGauge._checkpoint():

```
# LiquidityGauge._checkpoint()
r.integral_inv_supply = self.integral_inv_supply
if block.timestamp > r.integral_inv_supply.t:
    r.integral_inv_supply.v += unsafe_div(
        10**36 * (block.timestamp - r.integral_inv_supply.t),
        erc4626.erc20.totalSupply  # ← may be 0
    )
    r.integral_inv_supply.t = block.timestamp
```

LiquidityGauge.vy#L146

If totalSupply() == 0, unsafe_div returns 0, and v is not increased. The entire d_reward is discarded: extra rewards accumulate in the contract but are not credited to anyone and can't be claimed.

Recommendation

We recommend checking totalSupply() > 0 before unsafe_div, or providing an admin-only claim function to recover unaccounted rewards if needed.

H-3	Infinitely Locked VotingEscrow Positions Are Not Accounted During Voting in GaugeController		
Severity	High	**Status**	Fixed in 4378752a

Description

In VotingEscrow, an infinite lock sets a constant vote bias equal to the deposit and a slope of 0. However, in GaugeController.vote_for_gauge_weights(), vote bias is recalculated as slope * dt, ignoring the constant bias from an infinite lock:

```
# Prepare slopes and biases in memory
old_slope: VotedSlope = self.vote_user_slopes[msg.sender][_gauge_addr]
old_dt: uint256 = max(old_slope.end, block.timestamp) - block.timestamp
old_bias: uint256 = old_slope.slope * old_dt
new_slope: VotedSlope = VotedSlope(
    slope = slope * _user_weight // 10000,  # slope = 0, so new_slope.slope = 0
    power = _user_weight,
    end = lock_end
)
new_dt: uint256 = lock_end - block.timestamp #
new_bias: uint256 = new_slope.slope * new_dt # new_bias = 0
```

GaugeController.vy#L235

As a result, votes from users with an infinite lock are ignored in gauge voting.

Recommendation

We recommend updating the new_bias calculation to support infinite locks.

H-4	Flash Loan Attack on Gauge Reward Distribution via get_adjustment() Manipulation		
Severity	High	Status	Fixed in a97447e0

Description

In the GaugeController._checkpoint_gauge() function, rewards are minted and the amount allocated to each gauge is computed for users to later claim. The gauge's adjusted weight (aw) is calculated based on Gauge(gauge).get_adjustment(), which internally evaluates LP_TOKEN.balanceOf(self) / LP_TOKEN.totalSupply(). The higher this ratio, the greater the share of newly minted rewards allocated to that gauge (as aw / aw_sum). This value can be manipulated using a flash loan, since deposit() and withdraw() can be called within a single transaction. An attacker can temporarily inflate the aw of their gauge to increase its reward share.

Example:

Assume there are two gauges, gauge1 and gauge2, with equal vote weights and 90% of the LP token total supply staked in each.

An attacker flash-loans assets from gauge1's pool, mints a large amount of LP tokens, deposits them into gauge1, and calls GC.checkpoint(gauge1) to register a much higher aw (due to the temporary spike in balanceOf(self)). In the same transaction, the attacker withdraws and repays the flash loan.

At this point, GaugeController holds an inflated aw for gauge1, increasing its reward ratio (aw / aw_sum). The attacker then waits (e.g., one day), allowing time-based rewards to accumulate under the manipulated weight. When Gauge1.claim() is called, rewards are distributed accordingly, granting the attacker a significantly larger share.

In this scenario, waiting one day can earn the attacker an extra 1.5% of total rewards. This figure grows with longer wait times or lower initial LP stake ratios in the targeted gauge.

Additional note:

An attacker could also use flash loans to increase the LP_TOKEN.totalSupply() in other gauges, reducing their adjustment and thereby their reward share. This doesn't boost rewards for gauge1, but acts as a denial-of-service (DoS) vector. However, Curve pool fees might outweigh the gains from such an attack, making the economic incentive unclear. Moreover, the attack is more effective in low-activity gauges, where the manipulated state can persist longer.

An example of the test was provided in the chat with the client.

Recommendation

We recommend either prohibiting unstaking in the same transaction or always calculating balanceOf()/totalSupply() as equal to 1.

Client's Commentary:

Client: *The issue is different from what is stated, but the test correctly covers TWO issues:*

1. *HIGH - checkpoints should be done AFTER Gauge token transfers, not before. Fixed in dc55c70b0a7192656ba2c8d54db093fd4a114647*
2. *Flashloan-assisted operations with LT combined with gauge checkpoint can temporarily disadvantage any gauge by making its adjustment small.*

The issue is even bigger: checkpoint must be done before transfers in LiquidityGauge but after in GaugeController.

2.3 Medium

M-1	Inflation Attack on LiquidityGauge		
Severity	Medium	**Status**	Fixed in a97447e0

Description

LiquidityGauge inherits from ERC-4626 and passes 0 as decimals offset (i.e., 1 virtual share).

```
# 0 - _DECIMALS_OFFSET
erc4626.__init__("YB Gauge: ..", "g(..)", lp_token, 0, "Just say no", "to EIP712")
```

LiquidityGauge.vy#L108

For 1 virtual share, an attacker can still perform a successful front-running attack on a victim, but to make a profit, they would need to intercept three victim deposits that do not exceed their own. Otherwise, the attack is more of a griefing - victims will be issued 0 shares for amounts less than the vault's balance.

Test example:

- append to tests/dao/test_gauge.py
- run with poetry run pytest tests/dao/test_gauge.py::test_inflation_attack -v -s

```
def test_inflation_attack(mock_lp, gauges, gc, yb, accounts, vote_for_gauges):
    hacker = accounts[0]
    victims = [accounts[1], accounts[2], accounts[3]]
    gauge = gauges[0]
    victimAmount = 10_000 * 10**18
    hackerAmout = victimAmount * 2

    hacker_balance_before = mock_lp.balanceOf(hacker)
    with boa.env.prank(hacker):
        gauge.mint(1, hacker)
        mock_lp.transfer(gauge.address, hackerAmout)

    for v in victims:
        with boa.env.prank(v):
            gauge.deposit(victimAmount, v)

    with boa.env.prank(hacker):
        gauge.redeem(gauge.balanceOf(hacker), hacker, hacker)

    print("Profit for hacker", mock_lp.balanceOf(hacker) - hacker_balance_before)
```

Recommendation

We recommend setting decimals offset higher, for example 3 (i.e., 1000 virtual shares).

Client's Commentary:

Client: *Let me try something different here. Instead, let's allow to have totalSupply of the gauge being either 0 or >= MIN_SHARES. This way, gauge forces the seed*

MixBytes: *After the fix: the hacker cannot use an inflation attack because the vault is restricted to a minimum of 1e12 shares.*

M-2	Incorrect Merging of Lock Ends		
Severity	Medium	**Status**	Fixed in 4378752a

Description

The `_merge_positions()` function in `VotingEscrow.vy` merges the locked amounts of two users, but does not update the `slope_changes` variable.

The issue is that `_ve_transfer_allowed()` allows merging positions with different lock end times (e.g., `UMAXTIME` and `max_value(uint256)`):

`VotingEscrow.vy#L521-L527`

As a result, positions with different lock end times can be merged, which leads to incorrect vote calculations, since `slope_changes` will not reflect the actual change in slope after merging.

Recommendation

We recommend either prohibiting the merging of positions with different lock end times or implementing logic to correctly update `slope_changes` during the merge operation.

2.4 Low

L-1	Documentation and Code Inconsistencies		
Severity	Low	**Status**	Fixed in 4378752a

Description

Several inconsistencies between documentation and implementation, as well as unused or misleading code, were identified:

- contracts/dao/GaugeController.vy: The NewGaugeWeight event is declared but never emitted anywhere in the contract, so it is effectively dead code.
- contracts/dao/GaugeController.vy: The admin state variable is declared with a comment but never initialized or used; ownership checks use ownable.owner, making admin redundant or misdocumented.
- contracts/dao/VotingEscrow.vy:getPastVotes(): The natspec comment refers to a clock() configured to use block numbers, but the implementation uses timestamps exclusively. The documentation is misleading.
- contracts/dao/VotingEscrow.vy:increase_amount(): The @notice documentation states it deposits additional tokens for msg.sender without modifying unlock time, but the function signature allows specifying a different _for address. The docs do not mention this parameter.
- contracts/dao/CliffEscrow.vy:__init__(): The constructor parameter is named recepient (misspelled).
- contracts/Factory.vy:set_gauge_cotroller(): Function name is misspelled as set_gauge_cotroller but the event is SetGaugeController (missing 'n' in the function name).

Recommendation

We recommend reviewing and updating documentation, comments, and code to ensure consistency and clarity.

L-2	Missing Zero-Address Checks in Factory Constructor		
Severity	Low	**Status**	Fixed in 4378752a

Description

Only price_oracle_impl is validated; amm_impl and lt_impl are not. If either is zero, create_from_blueprint will revert, bricking the factory.
Factory.vy#L110

Recommendation

We recommend asserting amm_impl and lt_impl are not zero.

L-3	unlock_time Can Exceed 4-Year Cap in CliffEscrow		
Severity	Low	**Status**	Fixed in 4378752a

Description

Only a future check is enforced in the constructor; no maximum limit is set for unlock_time. This allows unlock_time to exceed the intended 4-year cap, breaking VotingEscrow assumptions.

Recommendation

We recommend enforcing a maximum unlock_time consistent with the 4-year cap used in VotingEscrow.

L-4	One-Step Ownership Transfer on Critical Contracts		
Severity	Low	**Status**	Acknowledged

Description

`Factory`, `LiquidityGauge`, and `GaugeController` allow instant admin change. This risks fund loss in case of a human error during ownership transfers.

Recommendation

We recommend using two-step ownership transfers.

Client's Commentary:

Contracts are going to be owned by the DAO which executes everything by voting. 2-step processes usually do not play well with that.

L-5	Incorrect Bound Checking in Factory.add_market()		
Severity	Low	**Status**	Fixed in 4378752a

Description

In Factory.vy:add_market():

```
i: uint256 = self.market_count
if i < MAX_MARKETS:
    self.market_count = i + 1
self.markets[i] = market
```

If self.market_count == MAX_MARKETS, the code will overwrite the last market, which is unintended.

Recommendation

We recommend checking self.market_count < MAX_MARKETS.

3. About MixBytes

MixBytes is a leading provider of smart contract audit and research services, helping blockchain projects enhance security and reliability. Since its inception, MixBytes has been committed to safeguarding the Web3 ecosystem by delivering rigorous security assessments and cutting-edge research tailored to DeFi projects.

Our team comprises highly skilled engineers, security experts, and blockchain researchers with deep expertise in formal verification, smart contract auditing, and protocol research. With proven experience in Web3, MixBytes combines in-depth technical knowledge with a proactive security-first approach.

Why MixBytes

- **Proven Track Record:** Trusted by top-tier blockchain projects like Lido, Aave, Curve, and others, MixBytes has successfully audited and secured billions in digital assets.
- **Technical Expertise:** Our auditors and researchers hold advanced degrees in cryptography, cybersecurity, and distributed systems.
- **Innovative Research:** Our team actively contributes to blockchain security research, sharing knowledge with the community.

Our Services

- **Smart Contract Audits:** A meticulous security assessment of DeFi protocols to prevent vulnerabilities before deployment.
- **Blockchain Research:** In-depth technical research and security modeling for Web3 projects.
- **Custom Security Solutions:** Tailored security frameworks for complex decentralized applications and blockchain ecosystems.

MixBytes is dedicated to securing the future of blockchain technology by delivering unparalleled security expertise and research-driven solutions. Whether you are launching a DeFi protocol or developing an innovative dApp, we are your trusted security partner.

Contact Information

https://mixbytes.io/

https://github.com/mixbytes/audits_public

hello@mixbytes.io

https://x.com/mixbytes