VITAL BLOCK

Security Assessment

YIELD LEND

Vital Block Verified On Dec 12th, 2023

@Vital-Block

@VB_Audit

info@vitalblock.org

www.vitalblock.org

PREPARED FOR:

YIELD LEND

INTRODUCTION

Auditing Firm	**VITAL BLOCK SECURITY**
Client Firm	**YIELD LEND**
Methodology	**Automated Analysis, Manual Code Review**
Language	**Solidity**
Contract's Deployed. Reviewed & Audited	AaveOracle:0x0B9252d63db44eFa7f18911Ee2259cB40d0c2965 ACLManager:0x791946b3EF71E433aA1c4Fe7757bE11E2315d7Ce AToken:0x9b0379E8527E2a7439e92EfCDdce3719EeDA69AB DelegationAwareAToken:0xD7f4CdA790967327C36495eBc32dAF5666D1326b EmissionManager:0x4e1610187930Eb87238D2ec50029bE9012bFeaeE IncentivesProxy:0x56949a55b166404Fe0F6595fed1b54EA5C29137C IncentivesV2-Implementation:0x6e6Ac3b3b2c806C963aebB50D323f680bBc65246 Pool-Implementation:0x177a56705cE9A043cfD330eF479c4c37Af3E584C Pool-Proxy:0x35297537A1F8C67D3F7ab017A303F746982B3031 PoolAddressesProvider:0xFdf9DD6542E37B56Cdf59975246d5aEae7Aa832f PoolAddressesProviderRegistry:0x5a2A58CCc12B5f8ce1cea9f87479Bb30A1522775 PoolConfigurator-Implementation:0x87f34034c6452A68782B920d26A3509db4b1f28D PoolConfigurator-Proxy:0x47daA198BE7fea6d2032d2b9033D66b6CCA8C0A9 PoolDataProvider:0x43A5803c5f1Cb6241858669ad6F63fe5B3882434 ReservesSetupHelper:0x0bC60d5c371b4EF53dE86f9D3057f424a5fec0A8 ReserveStrategy-rateStrategyStableOne:0x37E69D754043E88FBca0C5C70618BbaF885241d1 ReserveStrategy-rateStrategyStableTwo:0x1520F9eD3eD3e7727E24645694612D5061BBCF27 ReserveStrategy-rateStrategyVolatileOne:0xaA0E84aa28492B0F9bC25d712395d738304a81b8 StableDebtToken:0xB70a62c2c4F1Dc03BfbaF6247EB213690292B8f3 UiIncentiveDataProviderV3:0x8d100Cb94CE4D15281043dA7e553df2148d76CE9 UiPoolDataProviderV3:0xACF2DD8C48BFf028b042bdF5F2Afb92Bac845D55 VariableDebtToken:0x24ba23Ccc0Fda941C8658A22f2694a01D252CD04 WalletBalanceProvider:0x5bbf658C38139865F5BdC12575453b3F6Dfaaba3 WrappedTokenGatewayV3:0xF68FfC771FD899dF13133F8A78EdddBAB78c253D
Blockchain	**BASE NETWORK**
Centralization	**Active ownership**
Website	**https://yieldlend.xyz**
Discord	**https://discord.yieldlend.xyz/**
Twitter	**https://twitter.com/yieldlend**
GitHub	**https://github.com/yieldlend/gov**
Prelim Report Date	**DECEMBER 11, 2023**
Final Report Date	**DECEMBER 12, 2023**

Verify the authenticity of this report on our GitHub Repo: https://www.github.com/vital-block

TABLE OF CONTENTS

Document Properties

Client	Yield Lend
Title	Smart Contract Audit Report
Target	Yield Lend
Version	1.0
Author	Akhmetshin Marat
Auditors	Akhmetshin Marat, James BK
Reviewed by	Dima Meru
Approved by	Prince Mitchell
Classification	Public

Version Info

Version	Date	Author(s)	Description
1.0	December 8, 2022	James BK	Final Release
1.0-AP	December 12, 2022	James BK	Release Candidate

Contact

For more information about this document and its contents, please contact Vital Block Security Inc.

Name	Akhmetshin Marat
Phone	+44 7944 248057
Email	info@vitalblock.org

In the following, we show the specific pull request and the commit hash value used in this audit.

- https://github.com/yieldlend/gov/tree/master/contracts (Y50PH590)
- https://github.com/yieldlend/gov/tree/master (33RTD778)

About Vital Block Security

Vital Block Security provides professional, thorough, fast, and easy-to-understand smart contract security audit. We do in-depth and penetrative static, manual, automated, and intelligent analysis of the smart contract. Some of our automated scans include tools like ConsenSys MythX, Mythril, Slither, Surya. We can audit custom smart contracts, DApps, NFTs, etc (including the service of smart contract auditing). We are reachable at Telegram (https://t.me/vital_block), Twitter (http://twitter.com/Vb_Audit), or Email (info@vitalblock.org).

Table 1.2: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Medium*	*Low*
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

Methodology (1)

To standardize the evaluation, we define the following terminology based on the OWASP Risk Rating Methodology [4]:

- Likelihood represents how likely a particular vulnerability is to be uncovered and exploited in the wild;
- Impact measures the technical loss and business damage of a successful attack;
- Severity demonstrates the overall criticality of the risk.

SCOPE OF WORK

Vital Block was consulted by YIELD LEND to conduct the smart contract audit of its. Sol source code.
The audit scope of work is strictly limited to mentioned .SOL file only:

○ **YIELDLEND.Sol**

External contracts and/or interfaces dependencies are not checked due to being out of scope.

Verify audited contract's contract address and deployed link below:

Public Contract Link
https://basescan.org/address/0x0B9252d63cb44eFa7f18911Ee2259cB40d0c2965
https://basescan.org/address/0x791946b3EF71E433aA1c4Fe7757bE11E2315d7Ce
https://basescan.org/address/0x9b0379E8527E2a7439e92EfCDdce3719EeDA69AB
https://basescan.org/address/0xD7f4CdA790967327C36495eBc32dAF5666D1326b
https://basescan.org/address/0x4e1610187930Eb87238D2ec50029bE9012bFeaeE
https://basescan.org/address/0x56949a55b166404Fe0F6595fed1b54EA5C29137C
https://basescan.org/address/0x6e6Ac3b3b2c806C963aebB50D323f680bBc65246
https://basescan.org/address/0x177a56705cE9A043cfD330eF479c4c37Af3E584C
https://basescan.org/address/0x35297537A1F8C67D3F7ab017A303F746982B3031
https://basescan.org/address/0xFdf9DD6542E37B56Ccf59975246d5aEae7Aa832f
https://basescan.org/address/0x5a2A58CCc12B5f8ce1cea9f87479Bb30A1522775
https://basescan.org/address/0x87f34034c6452A68782B920d26A3509db4b1f28D
https://basescan.org/address/0x47daA198BE7fea6d2032d2b9033D66b6CCA8C0A9
https://basescan.org/address/0x43A5803c5f1Cb6241858669ad6F63fe5B3882434
https://basescan.org/address/0x0bC60d5c371b4EF53dE86f9D3057f424a5fec0A8
https://basescan.org/address/0x37E69D754043E88FBca0C5C70618BbaF885241d1
https://basescan.org/address/0x1520F9eD3eD3e7727E24645694612D5061BBCF27
https://basescan.org/address/0xaA0E84aa28492B0F9bC25d712395d738304a81b8
https://basescan.org/address/0xB70a62c2c4F1Dc03BfbaF6247EB213690292B8f3
https://basescan.org/address/0x8d100Cb94CE4D15281043dA7e553df2148d76CE9
https://basescan.org/address/0xACF2DD8C48BFf028b042bcF5F2Afb92Bac845D55
https://basescan.org/address/0x24ba23Ccc0Fda941C8658A22f2694a01D252CD04
https://basescan.org/address/0x5bbf658C38139865F5BdC12575453b3F6Dfaaba3
https://basescan.org/address/0xF68FfC771FD899dF13133F8A78EddcBAB78c253D

AUDIT METHODOLOGY

Smart contract audits are conducted using a set of standards and procedures. Mutual collaboration is essential to performing an effective smart contract audit. Here's a brief overview of Vital Block auditing process and methodology:

CONNECT

- **The onboarding team gathers source codes, and specifications to make sure we understand the size, and scope of the smart contract audit.**

AUDIT

- **Automated analysis is performed to identify common contract vulnerabilities. We may use the following third-party frameworks and dependencies to perform the automated analysis:**
  - **Remix IDE Developer Tool**
  - **Open Zeppelin Code Analyzer**
  - **SWC Vulnerabilities Registry**
  - **DEX Dependencies, e.g., Pancakeswap, Uniswap**
- **Simulations are performed to identify centralized exploits causing contract and/or trade locks.**
- **A manual line-by-line analysis is performed to identify contract issues and centralized privileges. We may inspect below mentioned common contract vulnerabilities, and centralized exploits:**

Centralized Exploits	○ **Token Supply Manipulation** ○ **Access Control and Authorization** ○ **Assets Manipulation** ○ **Ownership Control** ○ **Liquidity Access** ○ **Stop and Pause Trading** ○ **Ownable Library Verification**

Common Contract Vulnerabilities	○ Integer Overflow ○ Lack of Arbitrary limits ○ Incorrect Inheritance Order ○ Typographical Errors ○ Requirement Violation ○ Gas Optimization ○ Coding Style Violations ○ Re-entrancy ○ Third-Party Dependencies ○ Potential Sandwich Attacks ○ Irrelevant Codes ○ Divide before multiply ○ Conformance to Solidity Naming Guides ○ Compiler Specific Warnings ○ Language Specific Warnings

REPORT

- The auditing team provides a preliminary report specifying all the checks which have been performed and the findings thereof.
- The client's development team reviews the report and makes amendments to the codes.
- The auditing team provides the final comprehensive report with open and unresolved issues.

PUBLISH

- The client may use the audit report internally or disclose it publicly.

It is important to note that there is no pass or fail in the audit, it is recommended to view the audit as an unbiased assessment of the safety of solidity codes.

Table 1.0 The Full Audit Checklist

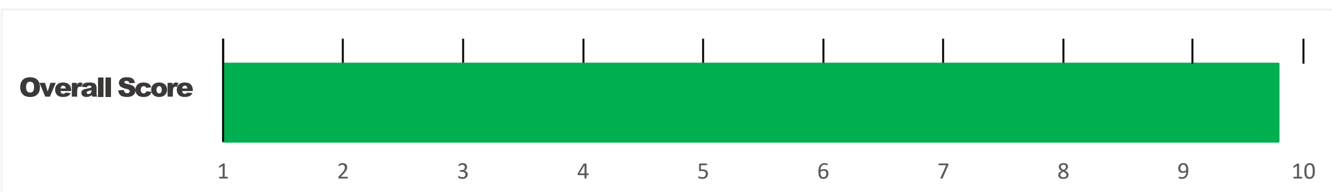
Category	Checklist Items
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	Money-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead Of Transfer
	Costly Loop
	(Unsafe) Use Of Untrusted Libraries
	(Unsafe) Use Of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses
Semantic Consistency Checks	Semantic Consistency Checks
Advanced DeFi Scrutiny	Business Logics Review
	Functionality Checks
	Authentication Management
	Access Control & Authorization
	Oracle Security
	Digital Asset Escrow
	Kill-Switch Mechanism
	Operation Trails & Event Generation
	ERC20 Idiosyncrasies Handling
	Frontend-Contract Integration
	Deployment Consistency
	Holistic Risk Management
Additional Recommendations	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

EXECUTIVE SUMMARY

Vital Block Security has performed the automated and manual analysis of the Yield Lend Sol code. The code was reviewed for common contract vulnerabilities and centralized exploits. Here's a quick audit summary:

Status	Critical !	Major "	Medium #	Minor $	Unknown %
Open	0	0	0	2	0
Acknowledged	1	0	0	0	0
Resolved	1	0	3	0	4
Noteworty onlyOwner Privileges	**Set Taxes and Ratios, Airdrop, Set Protection Settings, Set Reward Properties, Set Reflector Settings, Set Swap Settings, Set Pair and Router**				

YIELD LEND Smart contract has achieved the following score: 98.5



Please note that smart contracts deployed on blockchains aren't resistant to exploits, vulnerabilities and/or hacks. Blockchain and cryptography assets utilize new and emerging technologies. These technologies present a high level of ongoing risks. For a detailed understanding of risk severity, source code vulnerability, and audit limitations, kindly review the audit report thoroughly.

Please note that centralization privileges regardless of their inherited risk status - constitute an elevated impact on smart contract safety and security.

CENTRALIZED PRIVILEGES

Centralization risk is the most common cause of cryptography asset loss. When a smart contract has a privileged role, the risk related to centralization iselevated.

There are some well-intended reasons haveprivileged roles, such as:

- **Privileged roles can be granted the power to** pause() **the contract in case of an external attack.**
- **Privileged roles can use functions like**, include(), **and** exclude() **to add or remove wallets from fees, swap checks, and transaction limits. This is useful to run a presale and to list on an exchange.**

Authorizing privileged roles to externally-owned-account (EOA) is dangerous. Lately, centralization-related losses are increasing in frequency and magnitude.

- **The client can lower centralization-related risks by implementing below mentioned practices:**
- **Privileged role's private key must be carefully secured to avoid any potential hack.**
- **Privileged role should be shared by multi-signature (multi-sig) wallets.**
- **Authorized privilege can be locked in a contract, user voting, or community DAO can be introduced to unlock the privilege.**
- **Renouncing the contract ownership, and privileged roles.**
- **Remove functions with elevated centralization risk.**

Understand the project's initial asset distribution. Assets in the liquidity pair should be locked. Assets outside the liquidity pair should be locked with a release schedule.

RISK CATEGORIES

Smart contracts are generally designed to hold, approve, and transfer tokens. This makes them very tempting attack targets. A successful external attack may allow the external attacker to directly exploit. A successful centralization-related exploit may allow the privileged role to directly exploit. All risks which are identified in the audit report are categorized here for the reader to review:

Risk Type	Definition
Critical !	These risks could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.
Major "	These risks are hard to exploit but very important to fix, they carry an elevated risk of smart contract manipulation, which can lead to high-risk severity.
Medium #	These risks should be fixed, as they carry an inherent risk of future exploits, and hacks which may or may not impact the smart contract execution. Low-risk re-entrancy-related vulnerabilities should be fixed to deter exploits.
Minor $	These risks do not pose a considerable risk to the contract or those who interact with it. They are code-style violations and deviations from standard practices. They should be highlighted and fixed nonetheless.
Unknown %	These risks pose uncertain severity to the contract or those who interact with it. They should be fixed immediately to mitigate the risk uncertainty.

All statuses which are identified in the audit report are categorized here for the reader to review:

Status Type	Definition
Open	Risks are open.
Acknowledged	Risks are acknowledged, but not fixed.
Resolved	Risks are acknowledged and fixed.

AUDIT SCOPE | YIELD LEND

ID	Repo	Comment	File	SHM321 Checksum
YBY	contracts/yieldlend/gov	cC512486	BondingCurveSale.sol	6788099YIRHVSK853PKFMGHEF44309200KDHFCBUGIJN
YBI	contracts/yieldlend/gov	cC512486	BonusPool.sol	347520JHDB7549H22H3BVDIOETYUHF009JBIKBDI33BJ4
YBW	contracts/yieldlend/gov	cC512486	FeeDistributor.sol	1988Y73HUGFDINN353840NFMTEJER73649RGFIMDIDH
YBG	contracts/yieldlend/gov	cC512486	Epoch.sol	4438648TEOHBF6378309EHROECNEPOEJDNETE8EYEU3
YBL	contracts/yieldlend/gov	cC512486	StakingEmissions.sol	66390028765RVNKDBYFTGW553T2KOEHIUUJJIJE
YBA	contracts/yieldlend/gov	cC512486	StreamedVesting.sol	09825539BDYG543DVNKOMIKEBYRJUFHHFHJFIE333222
YBJ	contracts/yieldlend/gov	cC512486	VestedYieldLend.sol	8654RJVT3DWI865YK26437903JJDGGDHGWY6E
YBE	contracts/yieldlend/gov	cC512486	YieldLend.sol	7763888636TGYGFFTFHBETT66TFTCTVYBHBYT
YBP	Contracts/yieldlend/gov	cC512486	YieldLendTimelock.sol	88530486494YRHFTEICBGEIEGWTWYWUHEJEHEIE33U3
YBM	contracts/yieldlend/gov	cC512486	YieldLocker.sol	1209873KHJLKJNFJHGE98763990029774BCUHHDUU239
YBV	contracts/yieldlend/gov/interfaces	cC512486	IYieldLend.sol	23456UGFYUHE98756EFHJHE7654ESDFGHGERTYUJ3897
YBQ	contracts/yieldlend/gov/interfaces	cC512486	IWETH.sol	37889UHBIONEO7TYRDFGVBN5678939IJWSFVDYUHDCI
YBS	contracts/yieldlend/gov/interfaces	cC512486	IAggregatorV3Interface.sol	678903098TFHJKFCPOIUGFGHJKE9865ERGBEIVBHE8767
YBR	contracts/yieldlend/gov/interfaces	cC512480	IAerodromeRouter.sol	98765SDFGBNFCOI56789UIYHGGHEJDIUYTRDCVBN3459
YCD	contracts/yieldlend/gov/interfaces	cC512481	IStreamedVesting.sol	3348y9808hgtrusvnmu43100ejfojgfnut8496230hb574he
YHU	contracts/yieldlend/gov/interfaces	cC512481	IAerodromePool.sol	9864byf5f379eig28ffre64085jv1613251guhkdmue87
YGG	contracts/yieldlend/gov/interfaces	cC512481	IAerodromeFactory.sol	7ej2d8jg765tjfiowg538ij74dwftyv6478ij3gs820
YTR	contracts/yieldlend/gov/interfaces	cC512481	IAggregatorV3Interface.sol	864fr46de438hdguw903rfdcb246dbuhb2917enk

AUTOMATED ANALYSIS

Symbol	Definition
🛑	Function modifies state
💵	Function is payable
🔒	Function is internal
🔐	Function is private
❗	Function is important

| **YIELD LEND** | Interface | |||

| └ | totalSupply | External ❗ | | ❗ |NO❗ |

| └ | decimals | External ❗ | | ❗ |NO❗ |

| └ | symbol | External ❗ | | ❗ |NO❗ |

| └ | name | External ❗ | | ❗ |NO❗ |

| └ | getOwner | External ❗ | |NO❗ |

| └ | balanceOf | External ❗ | ❗ |NO❗ |

| └ | transfer | External ❗ | " ❗ 🛑 |NO❗ |

| └ | allowance | External ❗ | ❗ |NO❗ |

| └ | approve | External ❗ | " ❗ 🛑 |NO❗ |

| └ | transferFrom | External ❗ | " |NO❗ |

||||||

| **IFactoryV2** | Interface | |||

| └ | getPair | External ❗ | |NO❗ |

| └ | createPair | External ❗ | " |NO❗ |

||||||

| **IV2Pair** | Interface | |||

| └ | factory | External ❗ | |NO❗ |

| └ | getReserves | External ❗ | |NO❗ |

| └ | sync | External ❗ | " |NO❗ |

||||||

| **IRouter01** | Interface | |||

| └ | factory | External ❗ | |NO❗ |

| └ | ETH | External ❗ | |NO❗ |

| └ | addLiquidityETH | External ❗ | # |NO❗ |

| └ | addLiquidity | External ❗ | " |NO❗ |

| └ | swapExactAPTForTokens | External ❗ | # |NO❗ |

| └ | getAmountsOut | External ❗ | |NO❗ |

| └ | getAmountsIn | External ❗ | |NO❗ |

||||||

| **IRouter02** | Interface | IRouter01 |||

| └ | swapExactTokensForETHSupportingFeeOnTransferTokens | External ❗ | " |NO❗ |

| └ | swapExactETHForTokensSupportingFeeOnTransferTokens | External ❗ | # |NO❗ |

| └ | swapExactTokensForTokensSupportingFeeOnTransferTokens | External ❗ | " ❗ 🛑 |NO❗ |

| └ | swapExactTokensForTokens | External ❗ | " |NO❗ |

||||||

| **Protections** | Interface | |||

| └ | checkUser | External ❗ | " ❗ 🛑 |NO❗ |

| └ | setLaunch | External ❗ | " ❗ 🛑 |NO❗ |

| └ | setLpPair | External ❗ | " ❗ 🛑 |NO❗ |

| └ | YIELD | External ❗ | " |NO❗ |

| └ | removeSniper | External ❗ | " |NO❗ |

||||||

| **Cashier** | Interface | |||

| └ | setRewardsProperties | External ❗ | " |NO❗ |

| └ | tally | External ❗ | " |NO❗ |

| └ | load | External ❗ | # |NO❗ |

| └ | cashout | External ❗ | " |NO❗ |

| └ | giveMeWelfarePlease | External ❗ | " |NO❗ |

| └ | getTotalDistributed | External ❗ | |NO❗ |

| └ | getUserInfo | External ❗ | |NO❗ |

| └ | getUserRealizedRewards | External ❗ | |NO❗ |

| └ | getPendingRewards | External | |NO |

| └ | initialize | External | " |NO |

| └ | getCurrentReward | External | |NO |

||||||

| **SOL** | Implementation | SafeMath |||

| └ | <Constructor> | Public | # |NO |

| └ | transferOwner | External | " | onlyOwner |

| └ | renounceOwnership | External | " | NO!|

| └ | setOperator | Public | " |NO |

| └ | renounceOriginalDeployer | External | " |NO |

| └ | <Receive Ether> | External | # |NO |

| └ | totalSupply | External | |NO |

| └ | decimals | External | |NO |

| └ | symbol | External | |NO |

| └ | name | External | |NO |

| └ | getOwner | External | ! |NO |

| └ | balanceOf | Public | ! |NO |

| └ | allowance | External | ! |NO |

| └ | approve | External | " ! |NO |

| └ | _approve | Internal $ | " ||

| └ | approveContractContingency | Public | " ! | onlyOwner |

| └ | transfer | External | " |NO |

| └ | transferFrom | External | " |NO |

| └ | setNewRouter | External | " | onlyOwner |

| └ | setLpPair | External | " | onlyOwner |

| └ | setInitializers | External | " | onlyOwner |

| └ | isExcludedFromFees | External | |NO |

| └ | isExcludedFromDividends | External | |NO |

| └ | isExcludedFromProtection | External | |NO |

| └ | setDividendExcluded | Public | " | onlyOwner |

| └ | setExcludedFromFees | Public | " | onlyOwner |

OPTIMIZATIONS | YIELD LEND

ID	Title	Category	Status
STV	**Logarithm Refinement Optimization**	Gas Optimization	Acknowledged
SOP	**Checks Can Be Performed Earlier**	Gas Optimization	Acknowledged
SDP	**Unnecessary Use Of SafeMath**	Gas Optimization	Acknowledged
SWY	**Struct Optimization**	Gas Optimization	Acknowledged
SGT	**Unused State Variable**	Gas Optimization	Acknowledged

General Detectors

Public Functions Should be Declared External

Some functions in this contract should be declared as external in order to save gas

Attention Required

Missing Zero Address Validation

Some functions in this contract may not appropriately check for zero addresses being used.

Attention Required

Numeric Notation Best Practices

The numeric notation used in this contract is unconventional, possibly worsening the reading/debugging experience

Attention Required

- No compiler version inconsistencies found
- No unchecked call responses found
- No vulnerable self-destruct functions found
- No assertion vulnerabilities found
- No old solidity code found
- No external delegated calls found
- No external call dependency found
- No vulnerable authentication calls found
- No invalid character typos found
- No RTL characters found
- No dead code found
- No risky data allocation found
- No uninitialized state variables found
- No uninitialized storage variables found
- No vulnerable initialization functions found
- No risky data handling found
- No number accuracy bug found
- No out-of-range number vulnerability found
- No map data deletion vulnerabilities found
- No tautologies or contradictions found
- No faulty true/false values found
- No innacurate divisions found
- No redundant constructor calls found
- No vulnerable transfers found
- No vulnerable return values found
- No uninitialized local variables found
- No default function responses found
- No missing arithmetic events found
- No missing access control events found
- No redundant true/false comparisons found
- No state variables vulnerable through function calls found
- No buggy low-level calls found
- No expensive loops found
- No bad numeric notation practices found
- No missing constant declarations found
- No missing external function declarations found
- No vulnerable payable functions found
- No vulnerable message values found

Key Findings

Overall, these smart contracts are well-designed and engineered, though the implementation can be improved by resolving the identified issues (shown in Table 2.1), including 1 high-severity vulnerabilities, 3 medium-severity vulnerabilities, 4 low-severity vulnerabilities, and 2 informational recommendations.

Table 2.1: Key Yield Lend Audit Findings

ID	Severity	Title	Category	Status
YDL-001	Informational	Suggested immutable For Gas Efficiency	Coding Practice	Fixed
YDL-002	Medium	Proper And Consistent Collateral Enabling	Business Logic	Fixed
YDL-003	Low	Improvement on UserConfiguration::_getFirstAssetAsCollateralId()	Coding Practice	Fixed
YDL-004	Informational	Redundant State/Code Removal	Coding Practice	Fixed
YDL-005	High	Proper Asset Price in GenericLogic::calculateUserAccountData()	Business Logic	Fixed
YDL-006	Medium	Proper EMode Category Use in Pool::borrow()	Business Logic	Fixed
YDL-007	Low	Possible Underflow Avoidance in BorrowLogic And UserConfiguration	Coding Practices	Confirmed
YDL-008	Low	Consistent Reserve Cache Use in rebalanceStableBorrowRate()	Coding Practice	Fixed

Beside the identified issues, we emphasize that for any user-facing applications and services, it is always important to develop necessary risk-control mechanisms and make contingency plans, which may need to be exercised before the mainnet deployment. The risk-control mechanisms should kick in at the very moment when the contracts are being deployed on mainnet. Please refer to page 10 for details.

YDL-01 Key Findings

Category	Severity	Location	Status
Coding Practices	Low	Multiple Contracts	Informational

Description

In **Suggested Constant/Immutable Usages For Gas Efficiency**

```
contract PriceOracleSentinel is IPriceOracleSentinel {
  IPoolAddressesProvider public
  _addressesProvider; ISequencerOracle public
  _oracle;
  uint256 public _gracePeriod;
```

Description

Since version v0.8.10+, Solidity introduces the feature of declaring a state as immutable. An immutable state variable can only be assigned during contract creation, but will remain constant throughout the life-time of a deployed contract. The main benefit of declaring a state as immutable is that reading the state is significantly cheaper than reading from regular storage, since it is not stored in storage anymore. Instead, an immutable state will be directly inserted into the runtime code.

This feature is introduced based on the observation that the reading and writing of storage-based contract states are gas-expensive. Therefore, it is always preferred if we can reduce, if not eliminate, storage reading and writing as much as possible. Those state variables that are written only once are candidates of immutable states under the condition that each fits the pattern, i.e., "a constant, once assigned in the constructor, is read-only during the subsequent operation."

In the following, we show a number of key state variables defined in PriceOracleSentinel, including _addressesProvider, _oracle, and _gracePeriod. If there is no need to dynamically update these state variables, they can be declared as either constants or immutable for gas efficiency. In particular, the above three states can be defined as immutable.

Similarly, the _addressesProvider state in BondingCurveOracle and ACLManager can be defined as immutable for gas efficiency.

Recommendation

Revisit the state variable definition and make extensive use of constant/ immutable states.

YDL-02 Key Findings

Category	Severity	Target	Status
Business Logic	Medium	Pool	Fixed

Description

In **Improved Logic of Pool::_addReserveToList()**

```
function _addReserveToList(address asset) internal {
  uint256 reservesCount = _reservesCount;

  require(reservesCount < _maxNumberOfReserves, Errors.P_NO_MORE_RESERVES_ALLOWED); bool

  reserveAlreadyAdded = _reserves[asset].id != 0          _reservesList[0] == asset;

  if (!reserveAlreadyAdded) {
    for (uint8 i = 0; i <= reservesCount; i++) {
      if (_reservesList[i] == address(0)) {
        _reserves[asset].id = i;
        _reservesList[i] = asset;
        _reservesCount = reservesCount + 1;
```

Description

The Yield Lend protocol allows the governance to dynamically add new reserves into the protocol. To keep track of the list of active reserves, the protocol maintains the internal state `_reservesList`. While reviewing the accounting of active reserves, we notice the internal routine to add a new reserve needs to be improved.

To elaborate, we show Above the `_addReserveToList()` function. It implements a rather straight- forward logic in validating the new asset and then adding it into the internal `_reservesList`. It comes to our attention that the internal `for`-loop needs to terminate the execution once a vacant spot is located and populated. Note the current implementation will simply fill all available slots with the new reserve asset.

Recommendation

Revise the above `_addReserveToList()` function to proper add a new reserve asset.

YDL-03 Key Findings

Category	Severity	Target	Status
UserConfiguration	low	(UserConfiguration)	Fixed

Description

UserConfiguration::_getFirstAssetAsCollateralId()

```
function _getFirstAssetAsCollateralId(DataTypes.UserConfigurationMap memory self)
    internal
pure
returns (uint256)
                                                                {
unchecked {
    uint256 collateralData = self.data & COLLATERAL_MASK;
    uint256 firstCollateralPosition = collateralData & ~(collateralData - 1);
    uint256 id;
while ((firstCollateralPosition >>= 2) > 0) {
    id += 2;
}
return id / 2;
}
```

Description

The Yield lend protocol has a flexible mechanism to keep track of the configuration of current protocol users. This mechanism is mainly implemented in the UserConfiguration contract. In the process of reviewing this contract, we notice an internal helper function can be simplified

To elaborate, we show below this helper routine, i.e., _getFirstAssetAsCollateralId(). As the name indicates, this routine is designed to return the address of the first asset used as collateral by the user. It turns out the collateralData & ~(collateralData - 1) computation is unnecessary and the step size of 2 can be avoided as well.

Recommendation

Simplify the above routine as the follows:

```
function _getFirstAssetAsCollateralId(DataTypes.UserConfigurationMap memory
self)
   internal
   pure
   returns (uint256)
{
    uint256 collateralData= self.data & COLLATERAL_MASK;
   uint256 id;


  while ((collateralData>>= 2) > 0) {
   id += 1;
}
return id;
}
```

UserConfiguration::_getFirstAssetAsCollateralId()

YDL-04 Key Findings

Category	Severity	Target	Status
Coding Practices	Informational	Multiple Contracts	Fixed

Description

Redundant State/Code Removal

```
struct AccrueToTreasuryLocalVars {
  Uint256 prevTotalStableDebt;
  uint256 prevTotalVariableDebt;
  uint256 currTotalVariableDebt;
  uint256 avgStableRate;
  uint256 cumulatedStableInterest;
  uint256 totalDebtAccrued;
  uint256 amountToMint;
  uint40 stableSupplyUpdatedTimestamp;
}
```

Description

The Yield Lend protocol makes good use of a number of reference contracts, such as ERC20, SafeERC20, SafeMath, and Address, to facilitate its code implementation and organization. For example, the Pool smart contract has so far imported at least five reference contracts. However, we observe the inclusion of certain unused code or the presence of unnecessary redundancies that can be safely removed.

For example, if we examine closely the ReserveLogic library, there is an AccrueToTreasuryLocalVars structure with a number of member fields that are defined, but not used. Examples include the YieldStableRate and stableSupplyUpdatedTimestamp fields. Also, another structure UpdateInterestRatesLocalVars defines an unused member field YieldStableRate.

Recommendation

Consider the removal of the redundant state (or code) with a simplified, consistent implementation.

YDL-05 Key Findings

Category	Severity ●	Target	Status
Coding Practices	High	GenericLogic	FIXED

Description

Proper Asset Price in GenericLogic::calculateUserAccountData()

```
  function calculateUserAccountData(
    mapping(address => DataTypes.ReserveData) storage reservesData,
    mapping(uint256 => address) storage reserves,
    mapping(uint8 => DataTypes.EModeCategory) storage eModeCategories, Data
    Types.CalculateUserAccountDataParams memory params
    )
internal
view
returns (
  uint256,
  uint256,
  uint256,
  uint256,
  uint256,
  bool
)
                                                        {
if (params.userConfig.isEmpty()) {
  return (0, 0, 0, 0, type(uint256).max, false);
}
CalculateUserAccountDataVars memory vars;
```

Description

For any lending protocol, there is a need to reliably and accurately measure the borrower's debt position and provide necessary means to liquidate underwater positions. The Yield Lend protocol is no exception. While reviewing the implementation to measure the debt position, we notice the key function `calculateUserAccountData()` needs to be improved.

To illustrate, we show below this function. As the name indicates, the function is dedicated to calculate the user data across the reserves. For this end, it requires the total liquidity/collateral/bor- row balances in the base currency used by the price feed, as well as the average loan to value (LVT), the average liquidation ratio, and the health factor. However, it misuses the eModeAssetPrice as the price for each iterated reserve (lines 134-136), which leads to erroneous calculation of collateral value and borrow power. This issue is possibly introduced to support the eMode feature, but has been mistakenly used to consider all reserve assets to be part of the same eMode category.

.

Recommendation

Apply the right price oracle in the above `calculateUserAccountData()` routine to compute the user account data.

.

YDL-06 Key Findings

Category	Severity	Target	Status
Coding Practices	Medium	Pool	Fixed

Description

Proper EMode Category Use in Pool::borrow()

```
function borrow(
  address asset,
  uint256 amount
  ,
  uint256 interestRateMode,
  uint16 referralCode,
  address onBehalfOf
) external override { BorrowLogic.executeBorrow(
    _reserves,
    _reservesList,
    _eModeCategories,
    _usersConfig[onBehalfOf], DataTypes.ExecuteBorrowParams(
      asset,
      msg.sender,
      onBehalfOf,
      amount,
      interestRateMode,
      referralCode,
      true,
      _maxStableRateBorrowSizePercent,
      _reservesCount,
      _addressesProvider.getPriceOracle(),
      _usersEModeCategory[msg.sender],
      _addressesProvider.getPriceOracleSentinel()
    )
```

Description

The Yield Lend protocol has a nice feature credit delegation, which allows a credit delegator to delegate the credit of their account's position to a Lender. This feature requires proper accounting of delegation allowance and actual expenditure. While examining its implementation, we notice a key function borrow() does not properly follow the credit delegation logic.

To elaborate, we show Above this borrow() function. This is a core lending function and is used to borrow funds from the lending protocol. It comes to our attention that the encapsulated DataTypes. ExecuteBorrowParams parameters mistakenly uses _usersEModeCategory[msg.sender] as the user's eMode category. In the credit delegation situation, the real eMode category should be _usersEModeCategory[onBehalfOf].

Recommendation

Ensure the credit delegation feature is consistently honored in all aspects of the lending protocol.

YDL-07 Key Findings

Category	Severity ●	Target	Status
Coding Practices	low	BorrowLogic, UserConfiguration	Confirmed

Description

Possible Underflow Avoidance in BorrowLogic And UserConfiguration

```
function isUsingAsCollateralOne(DataTypes.UserConfigurationMap memory self)
  internal
  pure
  returns (bool)
{
uint256 collateralData = self.data & COLLATERAL_MASK;
return collateralData & (collateralData - 1) == 0;
}
```

Description

The Yield Lend protocol has established itself as one of the leading lending protocol. Within each lending proto-col, there is a constant need of accommodating various precision issues. SafeMath is a widely-used Solidity math library that is designed to support safe math operations by preventing common overflow or underflow issues when working with uint256 operands. Since the version 0.8.10, Solidity includes checked arithmetic operations by default, and this largely renders SafeMath unnecessary. While re-viewing arithmetic operations in current implementation, we notice occasions that may introduce unexpected overflows/underflows.

For example, if we examine the isUsingAsCollateralOne() function, it may revert if the current collateralData (line 120) is equal to 0. Another example is when the underlying asset of a reserve has an unusual decimal,

which may revert the following calculation of reserveCache.reserveConfiguration. getDecimals()-ReserveConfiguration.DEBT_CEILING_DECIMALS.

Note this calculation appears in a num- ber of routines. Its revert may bring in unnecessary frictions and cause issues for integration and composability.

Recommendation

Revise the above calculation to avoid the unnecessary overflows and under- flows.

YDL-08 Key Findings

Category	Severity	Target	Status
Coding Practices	low	BorrowLogic	Fixed

Description

Possible Underflow Avoidance in BorrowLogic And UserConfiguration

```
function executeSupply(
    mapping(address => DataTypes.ReserveData)
    storage reserves, mapping(uint256 => address)
    storage reservesList,          DataTypes.
    UserConfigurationMap storage userConfig,
    DataTypes.ExecuteSupplyParams memory params
) external {
  DataTypes.ReserveData storage reserve = reserves[params.asset];
  DataTypes.ReserveCache memory reserveCache = reserve.cache();
  reserve.updateState(reserveCache);
  ValidationLogic.validateSupply(reserveCache, params.amount);
  reserve.updateInterestRates(reserveCache, params.asset, params.amount, 0);
  ...
```

Description

For gas efficiency, the Yield Lend protocol is engineered with the reserve cache mechanism, which necessi-tates the common steps to be followed when operating with the reserve data in different scenarios, including the cache generation, update, and eventual persistence. However, our analysis shows cer- tain inconsistency in the reserve cache usages and the inconsistency needs to be resolved to avoid confusions and errors.

To elaborate, we show Above two functions `executeSupply()` and `rebalanceStableBorrowRate()`. These functions are self-explanatory and it comes to our attention that the first function updates the reserve cache before applying the validation logic while the second function validates the reserve cache before updating it. As mentioned earlier, this inconsistency may introduce issues when using the stale cache state for validation.

Recommendation

Revise the above functions to following a consistent approach to use the reserve cache mechanism.

Vulnerability Scan

REENTRANCY

No reentrancy risk found

Severity	Major
Confidence Parameter	Certain

Vulnerability Description

Mintable: More amount of the Yield Lend token can **NOT** be minted by a private wallet or contract. **(This is Essentially normal for most contracts)**

Scanning Line:

```
function yearnAgainAgain() public onlyOwner {
        require(!tradingActive, "Trading already active.");
        tradingActive = true;
        swapEnabled = true;
    }

    function setSwapEnabled(bool value) public onlyOwner {
        swapEnabled = value;
    }

    function setSwapTokensAtAmount(uint256 amount) public onlyOwner {
        require(
            amount >= (totalSupply() * 1) / 100000,
            "ERC20: Swap amount cannot be lower than 0.001% total supply."
        );
        require(
            amount <= (totalSupply() * 5) / 1000,
            "ERC20: Swap amount cannot be higher than 0.5% total supply."
        );
        swapTokensAtAmount = amount;
    }
```

Repository:

https://github.com/yieldlend/gov/tree/master/contracts

Additional Audited Files

BondingCurveOracle.sol
BonusPool.sol
BondingCurveSale.sol
Epoch.sol
FeeDistributor.sol
StakingEmissions.spl
StreamedVesting.sol
VestedYieldLend.sol
YieldLend.sol
YieldLendTimelock.sol
YieldLocker.sol

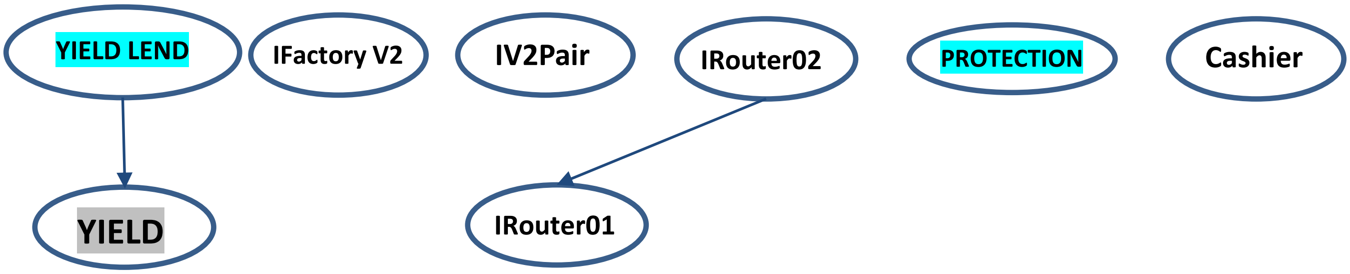
Contract Creator Address

0x823a37573f15bbdd950bbbb425fea29b41317510

Deployed Contracts:

```
Contract
AaveOracle: 0x0B9252d63cb44eFa7f18911Ee2259cB40d0c2965
ACLManager: 0x791946b3EF71E433aA1c4Fe7757bE11E2315d7Ce
AToken: 0x9b0379E8527E2a7439e92EfCDdce3719EeDA69AB
DelegationAwareAToken: 0xD7f4CdA790967327C36495eBc32dAF5666D1326b
EmissionManager: 0x4e1610187930Eb87238D2ec50029bE9012bFeaeE
IncentivesProxy: 0x56949a55b166404Fe0F6595fed1b54EA5C29137C
IncentivesV2-Implementation: 0x6e6Ac3b3b2c806C963aebB50D323f680bBc65246
Pool-Implementation: 0x177a56705cE9A043cfD330eF479c4c37Af3E584C
Pool-Proxy: 0x35297537A1F8C67D3F7ab017A303F746982B3031
PoolAddressesProvider: 0xFdf9DD6542E37B56Ccf59975246d5aEae7Aa832f
PoolConfigurator-Implementation: 0x87f34034c6452A68782B920d26A3509db4b1f28D
PoolConfigurator-Proxy: 0x47daA198BE7fea6d2032d2b9033D66b6CCA8C0A9
PoolDataProvider: 0x43A5803c5f1Cb6241858669ad6F63fe5B3882434
ReservesSetupHelper: 0x0bC60d5c371b4EF53dE86f9D3057f424a5fec0A8
ReserveStrategy-rateStrategyStableOn: 0x37E69D754043E88FBca0C5C70618BbaF885241d1
ReserveStrategy-rateStrategyStableTwo: 0x1520F9eD3eD3e7727E24645694612D5061BBCF27
ReserveStrategy-rateStrategyVolatileOne: 0xaA0E84aa28492B0F9bC25d712395d738304a81b8
StableDebtToken: 0xB70a62c2c4F1Dc03BfbaF6247EB213690292B8f3
UiIncentiveDataProviderV3: 0x8d100Cb94CE4D15281043dA7e553df2148d76CE9
UiPoolDataProviderV3: 0xACF2DD8C48BFf028b042bcF5F2Afb92Bac845D55
VariableDebtToken: 0x24ba23Ccc0Fda941C8658A22f2694a01D252CD04
WalletBalanceProvider: 0x5bbf658C38139865F5BdC12575453b3F6Dfaaba3
WrappedTokenGatewayV3: 0xF68FfC771FD899dF13133F8A78EddcBAB78c253D
```

INHERITANCE GRAPH



Identifier	Definition	Severity
CEN-12	Centralization privileges of YIELD LEND	Medium #

Vulnerability 0 : No important security issue detected.
Threat level: Low


```
import {IUniswapV2Pair} from "./interfaces/IUniswapV2Pair.sol";
import {IWETH} from "./interfaces/IWETH.sol";
import {IYieldLend} from "./interfaces/IYieldLend.sol";

contract YieldLend is IYieldLend, ERC20Burnable, Ownable {
    using SafeMath for uint256;

    IAerodromeRouter public immutable router;
    IWETH public immutable weth;

    address public pair;
    address public marketingWallet;
    address public constant deadAddress = address(0xdead);

    bool public tradingActive;
    bool public swapEnabled;
    bool private _swapping;

    uint256 public swapTokensAtAmount;

    uint256 public sellTotalFees;
    uint256 private _sellMarketingFee;
    uint256 private _sellBurnFee;
    uint256 private _sellLiquidityFee;

    uint256 private _tokensForMarketing;
```

PROJECT BASIC KNOWLEDGE

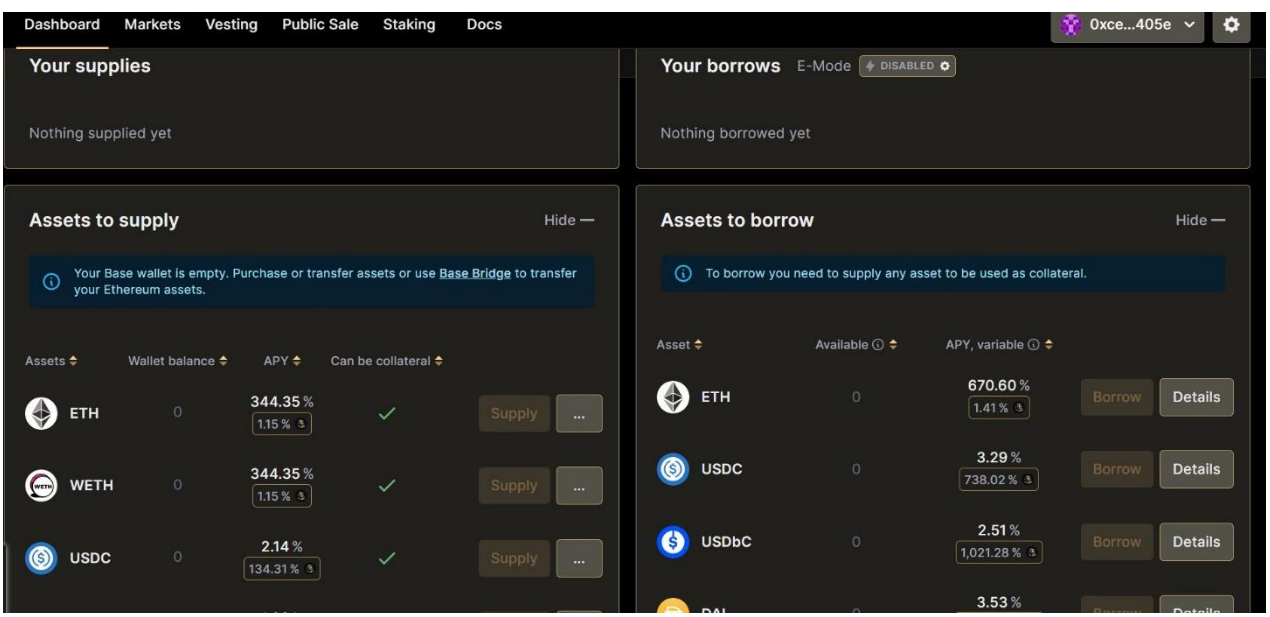
Yield Lend: is a lending protocol built on Base with veMeme tokenomics. YieldLend takes the best of Curve, Radiant Finance and Memecoin tokenomics, to create strong incentives that survive long-term growth.
With well designed tokenomics and a community driven fair launch, YieldLend strives to create a sustainable decentralized money market that can out-do other lending markets.

Yield lend builds on top of:veTokenomics of Curve/Solidly: Users can stake their YIELD tokens anywhere from 2 weeks to 4 years to have a vested interest in the ecosystem. The more tokens they stake, the bigger the share of rewards they earn.Reward Vesting Mechanics from Radiant Finance: Users that earn rewards from the protocol in various forms such as staking, farming etc. have their rewards vested. Rewards can be withdrawn early (with a penalty) or staked for 4 years (with a bonus).Tax-tokenomics from Memecoins: Users that trade the YIELD token pay a 5% tax on every sell that is used to contribute to token liquidity, increase token scarcity and generate revenue.Money market from Aave: Users provide liquidity into the money market which is based on Aave v3 to perform basic lending and borrowing. Providing liquidity not only earns incentives, but also generates revenue for the protocol.

PROJECT NAME: YIELD LEND
Ticker: YIELD

Chain/Standard: BASE NETWORK

The Yield Lend Platform Is Launched On Base Network



ISSUES CHECKING STATUS

	Issue Description	Checking Status
1.	Compiler errors.	PASSED
2.	Race Conditions and reentrancy. Cross-Function Race Conditions.	PASSED
3.	Possible Delay In Data Delivery.	PASSED
4.	Oracle calls.	PASSED
5.	Front Running.	PASSED
6.	Sol Dependency.	PASSED
7.	Integer Overflow And Underflow.	PASSED
8.	DoS with Revert.	PASSED
9.	Dos With Block Gas Limit.	PASSED
10.	Methods execution permissions.	PASSED
11.	Economy Model of the contract.	PASSED
12.	The Impact Of Exchange Rate On the solidity Logic.	PASSED
13.	Private use data leaks.	PASSED
14.	Malicious Event log.	PASSED
15.	Scoping and Declarations.	PASSED
16.	Uninitialized storage pointers.	PASSED
17.	Arithmetic accuracy.	PASSED
18.	Design Logic.	PASSED
19.	Cross-Function race Conditions	PASSED
20.	Save Upon solidity contract Implementation and Usage.	PASSED
21.	Fallback Function Security	PASSED

AUDIT RESULT

PASSED

Identifier	Definition	Severity
CEN-02	Initial asset distribution	Minor

All of the initially minted assets are sent to the contract deployer when deploying the contract. This is Normal for most deployer and/or contract owner .

```
function _swapTokensForETH(uint256 tokenAmount) internal {
    IAerodromeRouter.Route[] memory r = new IAerodromeRouter.Route[](1);
    IAerodromeRouter.Route memory route = IAerodromeRouter.Route({
        from: address(this),
        to: address(router.weth()),
        stable: false,
        factory: router.defaultFactory()
```

RECOMMENDATION

Project stakeholders should be consulted during the initial asset distribution process.

RECOMMENDATION

Deployer and/or contract owner private keys are secured carefully.

Please refer to PAGE-09 CENTRALIZED PRIVILEGES for a detailed understanding.

ALLEVIATION

The ARBITRUM EXCHANGE project team understands the centralization risk. Some functions are provided privileged access to ensure a good runtime behavior in the project

References

1 MITRE. CWE-1041: Use of Redundant Code. https://cwe.mitre.org/data/definitions/1041.html.

2 MITRE. CWE-1099: Inconsistent Naming Conventions for Identifiers. https://cwe.mitre.org/data/definitions/1099.html.

3 MITRE. CWE-561: Dead Code. https://cwe.mitre.org/data/definitions/561.html.

4 MITRE. CWE-563: Assignment to Variable without Use. https://cwe.mitre.org/data/definitions/563.html.

5 MITRE. CWE-663: Use of a Non-reentrant Function in a Concurrent Context. https://cwe.mitre.org/data/definitions/663.html.

6 MITRE. CWE-837: Improper Enforcement of a Single, Unique Action. https://cwe.mitre.org/data/definitions/837.html.

7 MITRE. CWE-841: Improper Enforcement of Behavioral Workflow. https://cwe.mitre.org/data/definitions/841.html.

8 MITRE. CWE CATEGORY: Bad Coding Practices. https://cwe.mitre.org/data/definitions/1006.html.

9 MITRE. CWE CATEGORY: Business Logic Errors. https://cwe.mitre.org/data/definitions/840.html.

10 MITRE. CWE CATEGORY: Concurrency. https://cwe.mitre.org/data/definitions/557.html.

11 MITRE. CWE VIEW: Development Concepts. https://cwe.mitre.org/data/definitions/699.html.

12 OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.

Identifier	Definition	Severity
COD-10	Third Party Dependencies	Minor

Smart contract is interacting with third party protocols e.g., Pancakeswap router, cashier contract, protections contract. The scope of the audit treats third party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised, and exploited. Moreover, upgrades in third parties can create severe impacts, e.g., increased transactional fees, deprecation of previous routers, etc.

RECOMMENDATION

Inspect and validate third party dependencies regularly, and mitigate severe impacts whenever necessary.

DISCLAIMERS

Vital Block Security provides the easy-to-understand audit of Solidity, Move and Raw source codes (commonly known as smart contracts).

The smart contract for this particular audit was analyzed for common contract vulnerabilities, and centralization exploits. This audit report makes no statements or warranties on the security of the code. This audit report does not provide any warranty or guarantee regarding the absolute bug-free nature of the smart contract analyzed, nor do they provide any indication of the client's business, business model or legal compliance. This audit report does not extend to the compiler layer, any other areas beyond the programming language, or other programming aspects that could present security risks. Cryptographic tokens are emergent technologies, they carry high levels of technical risks and uncertainty. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. This audit report could include false positives, false negatives, and other unpredictable results.

CONFIDENTIALITY

This report is subject to the terms and conditions (including without limitations, description of services, confidentiality, disclaimer and limitation of liability) outlined in the scope of the audit provided to the client. This report should not be transmitted, disclosed, referred to, or relied upon by any individual for any purpose without InterFi Network's prior written consent.

NO FINANCIAL ADVICE

This audit report does not indicate the endorsement of any particular project or team, nor guarantees its security. No third party should rely on the reports in any way, including to make any decisions to buy or sell a product, service or any other asset. The information provided in this report does not constitute investment advice, financial advice, trading advice, or any other sort of advice and you should not treat any of the report's content as such. This audit report should not be used in any way

to make decisions around investment or involvement. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort.

FOR AVOIDANCE OF DOUBT, SERVICES, INCLUDING ANY ASSOCIATED AUDIT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

TECHNICAL DISCLAIMER

ALL SERVICES, AUDIT REPORTS, SMART CONTRACT AUDITS, OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, VITAL BLOCK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO SERVICES, AUDIT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, VITAL BLOCK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM THE COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

WITHOUT LIMITING THE FOREGOING, VITAL BLOCK MAKES NO WARRANTY OF ANY KIND THAT ALL SERVICES, AUDIT REPORTS, SMART CONTRACT AUDITS, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET THE CLIENT'S OR ANY OTHER INDIVIDUAL'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE.

TIMELINESS OF CONTENT

The content contained in this audit report is subject to change without any prior notice. Vital Block does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following the publication.

LINKS TO OTHERWEBSITES

This audit report provides, through hypertext or other computer links, access to websites and social accounts operated by individuals other than Vital Block. Such hyperlinks are provided for your reference and convenience only and are the exclusive responsibility of such websites and social accounts owners. You agree that Vital block Security is not responsible for the content or operation of such websites and social accounts and that Vital Block shall have no liability to you or any other person or entity for the use of third-party websites and social accounts. You are solely responsible for determining the extent to which you may use any content at any other websites and social accounts to which you link from the report.

ABOUT VITAL BLOCK

Vital Block provides intelligent blockchain Security Solutions. We provide solidity and Raw Code Review, testing, and auditing services. We have Partnered with 15+ Crypto Launchpads, audited 50+ smart contracts, and analyzed 200,000+ code lines. We have worked on major public blockchains e.g., Ethereum, Binance, Cronos, Doge, Polygon, Avalanche, Metis, Fantom, Bitcoin Cash, Aptos, Oasis, etc.

Vital Block is Dedicated to Making Defi & Web3 A Safer Place. We are Powered by Security engineers, developers, UI experts, and blockchain enthusiasts. Our team currently consists of 5 core members, and 4+ casual contributors.

Website: https://Vitalblock.org

Email: info@vitalblock.org

GitHub: https://github.com/vital-block

Telegram (Engineering): https://t.me/vital_block

Telegram (Onboarding): https://t.me/vitalblock_cmo

vital-block

info@vitalblock.org

www.Vitalblock.org

Vital Block Dedicated to securing Public and Private Blockchain Ecosystem