CredShields

Smart Contract Audit

February 6, 2026 • CONFIDENTIAL

Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of Ardentis between January 7, 2026, and January 21, 2026. A retest was performed on February 4, 2026.

Author
Shashank (Co-founder, CredShields) shashank@CredShields.com

Reviewers
Aditya Dixit (Research Team Lead), Shreyas Koli(Auditor), Naman Jain (Auditor), Sanket Salavi (Auditor), Prasad Kuri (Auditor), Neel Shah (Auditor)

Prepared for
Ardentis

Table of Contents

1. Executive Summary ----------

Ardentis engaged CredShields to perform a smart contract audit from January 7, 2026, to January 21, 2026. During this timeframe, 35 vulnerabilities were identified. **A retest was performed on February 4, 2026, and all the bugs have been addressed.**

During the audit, 0 vulnerabilities were found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "Ardentis" and should be prioritized for remediation, and fortunately, none were found.

The table below shows the in-scope assets and a breakdown of findings by severity per asset. Section 2.3 contains more information on how severity is calculated.

Assets in Scope	Critical	High	Medium	Low	info	Gas	Σ
Smart Contract	0	0	1	20	10	4	**35**

Table: Vulnerabilities Per Asset in Scope

The CredShields team conducted the security audit to focus on identifying vulnerabilities in Smart Contract's scope during the testing window while abiding by the policies set forth by Ardentis's team.

State of Security

To maintain a robust security posture, it is essential to continuously review and improve upon current security processes. Utilizing CredShields' continuous audit feature allows both Ardentis's internal security and development teams to not only identify specific vulnerabilities but also gain a deeper understanding of the current security threat landscape.

To ensure that vulnerabilities are not introduced when new features are added, or code is refactored, we recommend conducting regular security assessments. Additionally, by analyzing the root cause of resolved vulnerabilities, the internal teams at Ardentis can implement both manual and automated procedures to eliminate entire classes of vulnerabilities in the future. By taking a proactive approach, Ardentis can future-proof its security posture and protect its assets.

2. The Methodology ----------

Ardentis engaged CredShields to perform a Smart Contract audit. The following sections cover how the engagement was put together and executed.

2.1 Preparation Phase

The CredShields team meticulously reviewed all provided documents and comments in the smart contract code to gain a thorough understanding of the contract's features and functionalities. They meticulously examined all functions and created a mind map to systematically identify potential security vulnerabilities, prioritizing those that were more critical and business-sensitive for the refactored code. To confirm their findings, the team deployed a self-hosted version of the smart contract and performed verifications and validations during the audit phase.

A testing window from January 7, 2026, to January 21, 2026, was agreed upon during the preparation phase.

2.1.1 Scope

During the preparation phase, the following scope for the engagement was agreed upon:

IN SCOPE ASSETS
Audited Scope: https://github.com/ardentis-lending/ardentis-contracts/tree/a6181f2e0e7edb0f94cbc240c080baa54f2847c9 **Retested Scope:** https://github.com/ardentis-lending/ardentis-contracts/tree/cd331af6bc604fd17dbba816b8ee919e8d8fa005

2.1.2 Documentation

Documentation was not required as the code was self-sufficient for understanding the project.

2.1.3 Audit Goals

CredShields employs a combination of in-house tools and thorough manual review processes to deliver comprehensive smart contract security audits. The majority of the audit involves manual inspection of the contract's source code, guided by OWASP's Smart Contract Security Weakness Enumeration (SCWE) framework and an extended, self-developed checklist built from industry best practices. The team focuses on deeply understanding the contract's core logic, designing targeted test cases, and assessing business logic for potential vulnerabilities across OWASP's identified weakness classes.

CredShields aligns its auditing methodology with the OWASP Smart Contract Security projects, including the Smart Contract Security Verification Standard (SCSVS), the Smart Contract Weakness Enumeration (SCWE), and the Smart Contract Secure Testing Guide (SCSTG). These frameworks, actively contributed to and co-developed by the CredShields team, aim to bring consistency, clarity, and depth to smart contract security assessments. By adhering to these OWASP standards, we ensure that each audit is performed against a transparent, community-driven, and technically robust baseline. This approach enables us to deliver structured, high-quality audits that address both common and complex smart contract vulnerabilities systematically.

2.2 Retesting Phase

Ardentis is actively partnering with CredShields to validate the remediations implemented towards the discovered vulnerabilities.

2.3 Vulnerability classification and severity

CredShields follows OWASP's Risk Rating Methodology to determine the risk associated with discovered vulnerabilities. This approach considers two factors - Likelihood and Impact - which are evaluated with three possible values - **Low**, **Medium**, and **High**, based on factors such as Threat agents, Vulnerability factors, and Technical and Business Impacts. The overall severity of the risk is calculated by combining the likelihood and impact estimates.

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	None	Low	Medium
		LOW	MEDIUM	HIGH
Likelihood				

Overall, the categories can be defined as described below -

1. Informational

We prioritize technical excellence and pay attention to detail in our coding practices. Our guidelines, standards, and best practices help ensure software stability and reliability. Informational vulnerabilities are opportunities for improvement and do not pose a direct risk to the contract. Code maintainers should use their own judgment on whether to address them.

2. Low

Low-risk vulnerabilities are those that either have a small impact or can't be exploited repeatedly or those the client considers insignificant based on their specific business circumstances.

3. **Medium**

   Medium-severity vulnerabilities are those caused by weak or flawed logic in the code and can lead to exfiltration or modification of private user information. These vulnerabilities can harm the client's reputation under certain conditions and should be fixed within a specified timeframe.

4. **High**

   High-severity vulnerabilities pose a significant risk to the Smart Contract and the organization. They can result in the loss of funds for some users, may or may not require specific conditions, and are more complex to exploit. These vulnerabilities can harm the client's reputation and should be fixed immediately.

5. **Critical**

   Critical issues are directly exploitable bugs or security vulnerabilities that do not require specific conditions. They often result in the loss of funds and Ether from Smart Contracts or users and put sensitive user information at risk of compromise or modification. The client's reputation and financial stability will be severely impacted if these issues are not addressed immediately.

6. **Gas**

   To address the risk and volatility of smart contracts and the use of gas as a method of payment, CredShields has introduced a "Gas" severity category. This category deals with optimizing code and refactoring to conserve gas.

2.4 CredShields staff

The following individual at CredShields managed this engagement and produced this report:

- Shashank, Co-founder CredShields shashank@CredShields.com

Please feel free to contact this individual with any questions or concerns you have about the engagement or this document.

3. Findings Summary ----------

This chapter contains the results of the security assessment. Findings are sorted by their severity and grouped by asset and OWASP SCWE classification. Each asset section includes a summary highlighting the key risks and observations. The table in the executive summary presents the total number of identified security vulnerabilities per asset, categorized by risk severity based on the OWASP Smart Contract Security Weakness Enumeration framework.

3.1 Findings Overview

3.1.1 Vulnerability Summary

During the security assessment, 35 security vulnerabilities were identified in the asset.

Vulnerability Title	Severity	Vulnerability Type	Status
Withdraw Function Does Not Enforce Minimum Remaining Supply Check	Medium	Inconsistent Accounting (SCWE-010)	Fixed
Residual Collateral Prevents Bad Debt Socialization	Low	Business Logic (SC03-LogicErrors)	Acknowledged
Flash Loan Enables ERC-4626 Collateral Price Collapse	Low	Business Logic (SC03-LogicErrors)	Acknowledged
Immediate Liquidations Can Trigger After Protocol Unpause	Low	Business Logic (SC03-LogicErrors)	Acknowledged
Liquidation Can Create Dust Borrow Positions For Ardentis Users	Low	Inconsistent Accounting (SCWE-010)	Acknowledged
Liquidators Can Leave Borrowers With Unhealthy Positions	Low	Inconsistent Accounting (SCWE-010)	Acknowledged
Same-Block Borrow and Repay Can Temporarily Block Liquidity	Low	Denial of Service (SCWE-087)	Acknowledged

Fees Accrue Even when Fee Recipient is the Zero Address	Low	Business Logic (SC03-LogicErrors)	Fixed
Manager Can Add Invalid Provider To Market	Low	Lack Of Input Validation (SC04-Lack Of Input Validation)	Fixed
Enabled IRM and LLTV Parameters Cannot Be Deactivated	Low	Missing functionality (SCWE-006)	Fixed
Lack of Slippage Protection in ArdentisVault	Low	Business Logic (SC03-LogicErrors)	Acknowledged
Attacker Can Grief Vault Depositors By Forcing Zero Share Mints	Low	Denial of Service (SCWE-087)	Acknowledged
Caller Can Receive Incorrect Ether Refund In Mint Flow	Low	Inconsistent Accounting (SCWE-010)	Fixed
Missing _disableInitialzers() call in constructor	Low	Insecure Upgradeable Proxy Design (SCWE-005)	Fixed
Use of Multiple Pragma Versions	Low	Missing Best Practices	Fixed
Missing Events in Important Functions	Low	Missing Best Practices	Fixed
Floating and Outdated Pragma	Low	Floating Pragma (SCWE-060)	Fixed
Provider Bypasses Ardentis Callback Mechanisms For Users	Low	Missing functionality (SCWE-006)	Not Fixed
Incorrect EmergencyWithdraw Event Amount For Native Asset (ETH/BNB)	Low	Wrong Event Emission	Fixed
Incorrect Event Emission in sellBNB()	Low	Wrong Event Emission	Fixed
Missing Price Feed Validation	Low	Lack of Input Validation (SC04-Lack Of Input Validation)	Fixed
Documentation Can Misrepresent Market Creation Permission Model For Users	Informational	Poor Governance Documentation (SCWE-015)	Not Fixed
Fee Transfers Can Be Failed Due To Rigid Ether Transfer Method	Informational	Missing Best Practices	Fixed
Protocol Cannot Recover Stuck Native Or Wrapped Tokens From Providers	Informational	Missing functionality (SCWE-006)	Fixed

Hardcoded Static Address	Informational	Hardcoded Constants (SCWE-008)	Acknowledged
Missing zero address validations	Informational	Missing Input Validation (SC04-Lack Of Input Validation)	Not Fixed
Documentation Can Misrepresent Supported Asset Tokens To Integrators	Informational	Documentation Mismatch	Not Fixed
Price Calculation Assumes Presence of Optional decimals() Function	Informational	Configuration Risk	Not Fixed
Use Ownable2Step	Informational	Missing Best Practices	Acknowledged
Lack of Access Control in reallocateTo Function	Informational	Access Control (SCWE-016)	Fixed
Oracle Decimal Mismatch Can Break Price Calculation For Markets	Informational	Lack of Input Validation (SC04-Lack Of Input Validation)	Not Fixed
Gas Optimization in Require/Revert Statements	Gas	Gas Optimization (SCWE-082)	Acknowledged
Cheaper conditional operators	Gas	Gas Optimization (SCWE-082)	Not Fixed
Custom error to save gas	Gas	Gas Optimization (SCWE-082)	Not Fixed
Gas Optimization in Increments	Gas	Gas optimization (SCWE-082)	Acknowledged

Table: Findings and Remediations

5. Bug Reports ----------

Bug ID #M001 [Fixed]

Withdraw Function Does Not Enforce Minimum Remaining Supply Check

Vulnerability Type
Inconsistent Accounting (SCWE-010)

Severity
Medium

Description
The supply() function enforces _checkSupplyAssets() after updating balances to ensure the supplier's remaining supply does not fall below the protocol-defined minimum. However, the withdraw() function does not perform the same check.

As a result, a user can withdraw assets in a way that leaves their remaining supply below the minimum allowed threshold, bypassing the invariant enforced during supply. This creates inconsistent behavior between supply and withdraw flows and allows states that the protocol explicitly prevents during supply.

Affected Code

- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis/Ardentis.sol#L321-L352

Impacts
Users can end up with supply balances below the intended minimum, leading to inconsistent accounting rules and weakening protocol assumptions around minimum liquidity or dust prevention. This can also complicate downstream logic that assumes _checkSupplyAssets() is always enforced symmetrically.

Remediation

It is recommended to apply _checkSupplyAssets(marketParams, onBehalf) in the withdraw() function after updating supply balances, similar to the supply() function, to ensure minimum remaining supply constraints are consistently enforced.

Retest

This issue has been fixed by calling _checkSupplyAssets in withdraw() function.

Bug ID #L001 [Acknowledged]

Residual Collateral Prevents Bad Debt Socialization

Vulnerability Type
Business Logic (SC03-LogicErrors)

Severity
Low

Description
The liquidate() function socializes bad debt only when the borrower's collateral balance becomes exactly zero. Because liquidators can control the liquidation size, a near-full partial liquidation can be performed that leaves a minimal amount of collateral (for example, 1 wei). In this situation, remaining borrow shares that are not covered by collateral are not socialized, even though they represent bad debt.

The post-liquidation health check does not reliably prevent this behavior, as _isHealthyAfterLiquidate() returns true when the remaining borrow amount is greater than or equal to the minimum loan size. This allows insolvent residual positions to remain open.

Affected Code
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis/Ardentis.sol#L541-L554
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis/Ardentis.sol#L793-L807

Impacts
Unrecoverable debt can persist in the system without being accounted for, leading to incorrect market accounting and increased insolvency risk. Liquidators with supply exposure are economically incentivized to avoid bad debt socialization.

Remediation
It is recommended that bad debt handling should not depend solely on the collateral balance reaching zero. The protocol should either enforce full liquidation when most collateral is seized or trigger bad debt socialization whenever the remaining borrow is not fully supported by the remaining collateral.

Retest

Client's Comment: Acknowledged.
Protocols like Aave and Morpho exhibit similar behavior where small residual or "dust" positions can remain after liquidation. This is a known and accepted design trade-off in industry-leading protocols.

Bug ID #L002 [Acknowledged]

Flash Loan Enables ERC-4626 Collateral Price Collapse

Vulnerability Type
Business Logic (SC03-LogicErrors)

Severity
Low

Description
Ardentis allows flash loans of non-blacklisted tokens without limits or fees. If an ERC-4626 share token is accepted as collateral and most of its supply is deposited into Ardentis, an attacker can use a flash loan to manipulate the share price.

By flash borrowing the ERC-4626 shares, the attacker can redeem them from the vault, draining the underlying assets and collapsing the share price. New shares can then be minted at the reduced price and used to repay the flash loan. This leaves the original collateral undervalued relative to the borrowed amount, resulting in protocol bad debt.

Affected Code
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis/Ardentis.sol#L589-L600

Impacts
Collateral value manipulation can lead to under-collateralized borrowing and bad debt, causing losses for liquidity providers and incorrect protocol solvency assumptions.

Remediation
It is recommended that at market creation, permanently lock a small portion of both collateral and loan token supply by sending it to address(0) or a dead address to ensure the full supply cannot be flash borrowed. Additionally, restrict or carefully validate ERC-4626 tokens used as collateral.

Retest
Client's Comment : Acknowledged.
Flash loans will be permitted only for standard ERC-20 tokens. To enforce this, we have introduced a flashLoanTokenBlacklist.

Bug ID #L003 [Acknowledged]

Immediate Liquidations Can Trigger After Protocol Unpause

Vulnerability Type
Business Logic (SC03-LogicErrors)

Severity
Low

Description

Ardentis allows the protocol to be paused and later unpaused by privileged roles. While the protocol is paused, borrowers cannot add collateral or repay their loans. During this time, positions may become unhealthy due to interest accrual or collateral price movements. When the protocol is unpaused, these positions become instantly liquidatable without giving borrowers any opportunity to recover their positions.

This creates a scenario where unpausing the protocol directly leads to immediate liquidations and borrower losses caused by conditions that could not be mitigated during the pause period.

Affected Code

- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis/Ardentis.sol#L816-L829

Impacts
Borrowers can suffer unexpected liquidations immediately after unpausing, resulting in avoidable losses and reduced user trust in protocol operations.

Remediation
It is recommended to introduce a grace period after unpausing during which liquidations are temporarily blocked, allowing borrowers sufficient time to add collateral or repay outstanding debt before liquidations resume.

Retest
Client's Comment: Acknowledged.
The pause and unpause mechanisms are triggered to protect the protocol from potential risks. While introducing a grace period after a unpause may appear fair to users who are unable to add collateral, disabling liquidations could expose the protocol to significant losses. To address this,

the protocol maintains an emergency fund that can be used to compensate affected users in exceptional circumstances.

Bug ID #L004 [Acknowledged]

Liquidation Can Create Dust Borrow Positions For Ardentis Users

Vulnerability Type
Inconsistent Accounting (SCWE-010)

Severity
Low

Description
The Ardentis protocol enforces a minimum loan size invariant to prevent the creation of economically insignificant dust positions. This invariant is applied in core user flows such as borrow and repay through internal checks that ensure a user's remaining borrow balance is either zero or greater than or equal to the configured minLoan threshold. These checks are intended to maintain consistency and economic soundness across the protocol. However, the liquidation flow follows a separate validation path that evaluates post-liquidation health using _isHealthyAfterLiquidate.

While this function verifies that the position meets health factor requirements after a partial liquidation, it does not enforce the same minimum loan size invariant. As a result, a partial liquidation can reduce a borrower's debt to a non-zero amount that is below minLoan without reverting, leaving the position in a dust state. The root cause is inconsistent invariant enforcement between liquidation logic and other borrowing-related flows.

Affected Code
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis/Ardentis.sol#L793C3-L807C4

Impacts
Liquidations can leave borrowers with dust borrow positions that are otherwise disallowed by the protocol, leading to inconsistent system states, increased operational complexity, and potential inefficiencies in managing or closing such positions.

Remediation

It is suggested to extend _isHealthyAfterLiquidate to enforce the minimum loan size invariant by ensuring that any remaining borrow balance after liquidation is either zero or greater than or equal to minLoan.

Retest

Client's Comment: Acknowledged.
We are aware of possible dust positions after liquidation.

Bug ID #L005 [Acknowledged]

Liquidators Can Leave Borrowers With Unhealthy Positions

Vulnerability Type
Inconsistent Accounting (SCWE-010)

Severity
Low

Description
After a liquidation, the protocol is expected to ensure that the borrower's remaining position is either fully closed or strictly healthy according to the same collateralization rules enforced during borrowing and withdrawals. This responsibility is handled by _isHealthyAfterLiquidate, which is invoked at the end of Ardentis.liquidate to validate the post-liquidation state.
However, _isHealthyAfterLiquidate first checks whether the remaining borrowed assets are greater than or equal to minLoan, and if so, it immediately returns true without validating collateralization via _isHealthy. This inverted check allows positions with sufficient remaining debt size but insufficient collateral value to be treated as healthy, bypassing the core health logic and allowing liquidation to finalize even though the position is still undercollateralized.

Affected Code

- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis/Ardentis.sol#L793C1-L807C4

Impacts
The user is partially liquidated and left with a non-zero amount of collateral and borrowed shares. Even though his remaining collateral is insufficient to support the outstanding debt, the liquidation succeeds because the remaining borrow amount exceeds minLoan, leaving an unhealthy position in the system and increasing the risk of unrecoverable bad debt for liquidity providers.

Remediation
It is recommended to validate collateral health first using _isHealthy and only apply the minLoan shortcut afterward to ensure no undercollateralized positions can persist after liquidation.

Retest
Client's Comment: Acknowledged.
Major lending platforms like Morpho don't perform health check after liquidation. They doesn't limit the liquidation size, as long as it reduces the overall risk of the protocol. In our case, health checks

are designed to make a position as healthy if its size falls below a minimum threshold. This is by design.

Bug ID #L006 [Acknowledged]

Same-Block Borrow and Repay Can Temporarily Block Liquidity

Vulnerability Type
Denial of Service (SCWE-087)

Severity
Low

Description
The protocol allows borrowing and repaying loan assets within the same block, while interest does not accrue intra-block. This enables a griefing attack where an attacker can temporarily consume all available liquidity without cost other than gas.

By front-running a legitimate borrow or withdraw transaction, an attacker can borrow all available liquidity using sufficient collateral, causing the honest transaction to revert due to insufficient liquidity. The attacker can then repay the borrow and withdraw their collateral within the same block, fully restoring their position.

Affected Code

- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis/Ardentis.sol#L357-L396
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis/Ardentis.sol#L399-L430

Impacts
Suppliers can be prevented from withdrawing their loan tokens, and borrowers can be blocked from accessing liquidity. This creates a low-cost, repeatable denial-of-service condition that negatively impacts protocol usability and reliability.

Remediation
It is recommended that restrict borrowing and repaying within the same block or enforce a minimal delay between these actions.

Retest

Client's Comment: Acknowledged.

We acknowledge that this scenario is theoretically possible; however, it results only in temporary, same-block liquidity usage with no lasting impact on the protocol or users. The behavior is purely griefing, economically irrational, and commonly accepted across major lending protocols. Introducing additional constraints to mitigate this edge case would add unnecessary complexity without meaningful security benefits.

Bug ID #L007 [Fixed]

Fees Accrue Even when Fee Recipient is the Zero Address

Vulnerability Type
Business Logic (SC03-LogicErrors)

Severity
Low

Description
The protocol allows the fee recipient to be set to address(0) using setFeeRecipient(). However, interest fees continue to accrue even when the fee recipient is the zero address. Since no valid recipient exists, the accrued fees cannot be claimed and remain stuck within the protocol's accounting.

This behavior may be intentional, but the implementation does not clearly distinguish between accruing fees for distribution and accruing fees with no recipient.

Affected Code

- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis/Ardentis.sol#L158C2-L164C4

Impacts
Accrued fees become unclaimable when the fee recipient is the zero address, leading to locked protocol revenue and unclear fee accounting.

Remediation
It is recommended that when feeRecipient is set to address(0), interest fee accrual should be skipped, or the protocol should explicitly store and later redistribute the accrued fees once feeRecipient is updated to a non-zero address.

Retest
This issue has been fixed by validating the zero address on feeRecipent.

Bug ID #L008 [Fixed]

Manager Can Add Invalid Provider To Market

Vulnerability Type
Lack Of Input Validation (SC04-Lack Of Input Validation)

Severity
Low

Description
The Ardentis.addProvider function allows a manager to register an external provider contract for a given market and token, enabling that provider to interact with borrowing, collateral, or liquidation flows. The intended design assumes that the provider's TOKEN() corresponds to either the loan token or collateral token defined in the market parameters. However, the function only checks that the token returned by IProvider(provider).TOKEN() is non-zero and that the market exists, without verifying that this token is actually part of the specified market. As a result, a provider associated with an unrelated token can be added to a market, creating inconsistent configuration and breaking assumptions in downstream logic that relies on providers being tied to valid market tokens.

Affected Code
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis/Ardentis.sol#L230

Impacts
A misconfigured or malicious manager can attach a provider for an unrelated token to an existing market, potentially causing unexpected reverts, blocked withdrawals, or incorrect provider-gated logic during borrow, supply, collateral withdrawal, or liquidation flows.

Remediation
It is recommended to ensure that the provider token matches either the loan token or the collateral token defined in the market parameters before registering the provider.

Retest
This issue has been fixed by validating tokens in providing markets.

Bug ID #L009 [Fixed]

Enabled IRM and LLTV Parameters Cannot Be Deactivated

Vulnerability Type
Missing functionality (SCWE-006)

Severity
Low

Description
Ardentis allows the MANAGER to enable Interest Rate Models (IRM) and Loan-to-Loan-Value (LLTV) parameters, but once enabled, there is no mechanism to disable or deactivate them. If an enabled IRM or LLTV parameter is later found to be faulty, misconfigured, or unsafe, the protocol has no way to promptly mitigate the risk.

This limits the protocol's ability to respond to parameter-level issues and increases operational risk during emergencies or misconfiguration events.

Affected Code

- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis/Ardentis.sol#L110C1-L129C4

Impacts
If an enabled IRM or LLTV parameter behaves incorrectly, the protocol cannot disable it, potentially exposing markets to incorrect interest calculations, unsafe borrowing limits, or systemic risk.

Remediation
It is recommended to add MANAGER controlled functions to disable previously enabled IRM and LLTV parameters, allowing the protocol to respond quickly to configuration errors or unexpected parameter behavior.

Retest
This issue has been fixed by introducing disableIrm() and disableLltv().

Bug ID #L010 [Acknowledged]

Lack of Slippage Protection in ArdentisVault

Vulnerability Type
Business Logic (SC03-LogicErrors)

Severity
Low

Description
The ArdentisVault contract allows users to interact with the vault by adding or removing liquidity through the deposit, mint, withdraw and redeem functions, which convert between underlying assets and vault shares according to the current exchange rate. These functions are intended to provide predictable value transfers based on the share-to-asset ratio at the time of execution. However, none of these entry points accept slippage parameters or enforce minimum expected amounts, meaning the execution always proceeds regardless of unfavorable rate changes. The root cause is the absence of user-specified bounds on assets or shares returned, despite the exchange rate being able to change over time due to interest accrual or realized bad debt.

Affected Code
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis-vault/ArdentisVault.sol#L423

Impacts
Users may receive fewer assets or fewer shares than expected when interacting with the vault, leading to unprotected value loss during periods of rapid exchange rate changes or adverse vault state transitions.

Remediation
It is recommended to add user-defined minimum output parameters to the deposit, mint, withdraw and redeem functions to ensure transactions revert when slippage exceeds acceptable thresholds.

Retest
Client's Comment: Acknowledged.

Bug ID #L011 [Acknowledged]

Attacker Can Grief Vault Depositors By Forcing Zero Share Mints

Vulnerability Type
Denial of Service (SCWE-087)

Severity
Low

Description
The Ardentis Vault is designed to convert between assets and shares using a deterministic share to asset exchange rate derived from total assets and total shares. This mechanism assumes sufficient initial liquidity and relies on a decimal offset to preserve precision during early vault usage. However, when the vault's underlying asset uses 18 decimals, the DECIMAL_OFFSET evaluates to zero, causing the share to asset conversion formulas to lose precision in low liquidity states. As a result, on a newly deployed vault with minimal initial deposits, an attacker can manipulate the vault's total assets before a victim deposit is processed, causing the victim's share calculation to round down to zero. The root cause is the combination of zero decimal offset and integer truncation in the share minting logic during early vault initialization.

Affected Code
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis-vault/ArdentisVault.sol#L103

Impacts
Users depositing into a newly deployed Ardentis Vault may receive zero or significantly fewer shares than expected due to precision loss, resulting in effective loss of deposited assets and enabling griefing attacks that undermine user trust and vault usability.

Remediation
It is recommended to ensure the vault cannot be initialized in a low liquidity state by enforcing a trusted first deposit or introducing a non-zero initial share supply to preserve precision.

Retest
Client's Comment: Acknowledged.
We acknowledge the plan to mitigate this via operational procedures (seeding initial liquidity) and documentation.

Bug ID #L012 [Fixed]

Caller Can Receive Incorrect Ether Refund In Mint Flow

Vulnerability Type
Inconsistent Accounting (SCWE-010)

Severity
Low

Description
The ETHProvider.mint function allows users to mint vault shares by sending ETH, where the contract first previews the required assets using IArdentisVault.previewMint and then performs the actual mint via IArdentisVault.mint. The intended behavior is to refund any excess ETH sent by the caller after minting is completed. However, the refund condition currently compares msg.value against previewAssets, which is only an estimate and not guaranteed to match the actual asset amount consumed by the mint call. Since IArdentisVault.mint returns the real amount of assets used, relying on the preview value introduces a mismatch between estimation and execution. The root cause is that the refund logic is not checked against the return value of mint, but against the earlier preview.

Affected Code
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/provider/ETHProvider.sol#L98

Impacts
Users send ETH to mint shares and receive an incorrect refund if the previewed assets differ from the actual assets consumed, resulting in either excess ETH being retained by the contract or an over-refund that breaks accounting assumptions.

Remediation
It is recommended to use the asset amount returned by IArdentisVault.mint as the reference for refund calculations instead of the previewed value.

Retest
This issue has been fixed using recommended remediation.

Bug ID #L013 [Fixed]

Missing _disableInitialzers() call in constructor

Vulnerability Type
Insecure Upgradeable Proxy Design (SCWE-005)

Severity
Low

Description
When using UUPSUpgradeable, the best practice is to call _disableInitizliers() in the constructor since the initializer function in the implementation contract can be called by anyone. The proxy's storage is separate from the implementation contract's storage. Directly initializing the implementation means you're not affecting the state that will be used when interacting via the proxy. This can lead to confusion or unexpected behavior if someone accidentally initializes the implementation contract.

Affected Code
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/ResilientOracle.sol#L36

Impacts
An uninitialized implementation contract can be taken over by an attacker.

Remediation
It is recommended to call _disableInitializers() in the constructor.

Eg:

```
constructor() {
  _disableInitializers();
}
```

Retest
This issue has been fixed by calling _disableInitializers() in the constructor.

Bug ID #L014 [Fixed]

Use of Multiple Pragma Versions

Vulnerability Type
Missing Best Practices

Severity
Low

Description
The contracts were found to be using multiple Solidity Compiler versions across different solidity files. This is not a good coding practice because different versions of the compiler have different caveats, breaking changes and introducing vulnerabilities.

Affected Code

- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/timelock/TimeLock.sol#L2
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/revenue/LendingRevenueDistributor.sol#L2
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/BoundValidator.sol#L2
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/ResilientOracle.sol#L2
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/OracleAdaptor.sol#L2

Impacts
Having different pragma versions across multiple contracts increases the chances of introducing vulnerabilities since each solidity version have their own set of issues and coding practices. Some major version upgrades may also break the contract logic if not handled properly.

Remediation
Instead of using different versions of the Solidity compiler with different bugs and security checks, it is better to use one version across all contracts.

Retest
This issue has been fixed.

Bug ID #L015 [Fixed]

Missing Events in Important Functions

Vulnerability Type
Missing Best Practices

Severity
Low

Description
Events are inheritable members of contracts. When you call them, they cause the arguments to be stored in the transaction's log—a special data structure in the blockchain. These logs are associated with the address of the contract which can then be used by developers and auditors to keep track of the transactions.

The contract was found to be missing these events on certain critical functions which would make it difficult or impossible to track these transactions off-chain.

Affected Code

- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/revenue/LendingRevenueDistributor.sol#L125C1-L131C1
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/timelock/TimeLock.sol#L36
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/revenue/LendingRevenueDistributor.sol#L125
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/revenue/LendingRevenueDistributor.sol#L133
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/revenue/LendingRevenueDistributor.sol#L141
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/liquidator/PublicLiquidator.sol#L441
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis-vault/ArdentisVault.sol#L217
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis-vault/ArdentisVault.sol#L342
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis-vault/ArdentisVault.sol#L348

- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/provider/ETHProvider.sol#L281
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/provider/ETHProvider.sol#L291
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/provider/BNBProvider.sol#L315
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/provider/BNBProvider.sol#L325

Impacts
Events are used to track the transactions off-chain and missing these events on critical functions makes it difficult to audit these logs if they're needed at a later stage.

Remediation
Consider emitting events for important functions to keep track of them.

Retest
This issue has been fixed.

Bug ID #L016 [Fixed]

Floating and Outdated Pragma

Vulnerability Type
Floating Pragma (SCWE-060)

Severity
Low

Description
Locking the pragma helps ensure that the contracts do not accidentally get deployed using an older version of the Solidity compiler affected by vulnerabilities.
The contract allowed floating or unlocked pragma to be used, i.e., >= 0.8.10. This allows the contracts to be compiled with all the solidity compiler versions above the limit specified. The following contracts were found to be affected -

Affected Code

- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/timelock/TimeLock.sol#L2
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/vault-allocator/VaultAllocator.sol#L2
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/revenue/LendingFeeRecipient.sol#L2
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/revenue/LendingRevenueDistributor.sol#L2
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis/Ardentis.sol#L3
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/BoundValidator.sol#L2
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/PTLinearDiscountOracle.sol#L2
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/PTLinearDiscountMarketOracle.sol#L2
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/ResilientOracle.sol#L2
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/OracleAdaptor.sol#L2

- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/liquidator/Liquidator.sol#L2
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/liquidator/PublicLiquidator.sol#L2
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/utils/BatchManagementUtils.sol#L2
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/interest-rate-model/FixedRateIrm.sol#L2
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/interest-rate-model/InterestRateModel.sol#L2
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis-vault/ArdentisVault.sol#L2
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis-vault/ArdentisVaultFactory.sol#L2
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis-vault/ArdentisVaultFactoryInternal.sol#L2
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/provider/ETHProvider.sol#L2
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/provider/BNBProvider.sol#L2

Impacts

If the smart contract gets compiled and deployed with an older or too recent version of the solidity compiler, there's a chance that it may get compromised due to the bugs present in the older versions or unidentified exploits in the new versions.
Incompatibility issues may also arise if the contract code does not support features in other compiler versions, therefore, breaking the logic.
The likelihood of exploitation is low.

Remediation

Keep the compiler versions consistent in all the smart contract files. Do not allow floating pragmas anywhere. It is suggested to use the 0.8.32 pragma version
Reference: https://scs.owasp.org/SCWE/SCSVS-CODE/SCWE-060/

Retest

This issue has been fixed.

Bug ID #L017 [Not Fixed]

Provider Bypasses Ardentis Callback Mechanisms For Users

Vulnerability Type
Missing functionality (SCWE-006)

Severity
Low

Description
The Ardentis core contract supports user-defined callback mechanisms in specific flows to enable advanced integrations and composability. In particular, the repay function can invoke a callback on the caller via onArdentisRepay, and the supplyCollateral function can invoke onArdentisSupplyCollateral, allowing users to execute custom logic after these operations complete. These callbacks are triggered when users interact directly with the Ardentis contract. However, when users interact through the ETHProvider or BNBProvider, these callbacks are no longer reachable because the provider contracts act as the direct caller to Ardentis and do not forward or re-expose the callback logic to the original user. In addition, Ardentis enforces provider-only collateral supply for certain assets through a restriction that prevents direct user interaction when a provider is registered, making the provider path mandatory. The root cause is that the provider contracts do not implement or forward the Ardentis callback interfaces despite being the required interaction layer for ETH and BNB markets.

Affected Code
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/provider/ETHProvider.sol#L201

Impacts
Users and integrators relying on onArdentisRepay or onArdentisSupplyCollateral callbacks cannot use these hooks when interacting with ETH or BNB markets through the provider contracts, reducing composability and potentially breaking existing integrations or automated workflows.

Remediation
It is recommended to extend the ETHProvider and BNBProvider contracts to forward or proxy the Ardentis callback mechanisms to the original caller, or explicitly document the lack of callback support when interacting through providers.

Retest

This issue has not been fixed.

Bug ID #L018 [Fixed]

Incorrect EmergencyWithdraw Event Amount For Native Asset (ETH/BNB)

Vulnerability Type
Wrong Event Emission

Severity
Low

Description
In `emergencyWithdraw()`, when `assets[i] == address(0)`, the contract sends `address(this).balance` to `msg.sender` and then emits `EmergencyWithdrawn` with `amount = address(this).balance`. At this point the balance is zero, so the event logs an incorrect amount (0) instead of the amount actually withdrawn.

Affected Code

- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/revenue/LendingRevenueDistributor.sol#L157

Impacts
Off-chain monitoring, accounting, and incident response relying on the event will record a zero amount, creating discrepancies between on-chain transfers and logs. - Can hinder audits and reconciliation of emergency operations.

Remediation
It is recommended to store the `ETH` amount to be withdrawn in a local variable prior to the transfer and emit that value in the event.

Retest
This issue has been fixed.

Bug ID #L019 [Fixed]

Incorrect Event Emission in sellBNB()

Vulnerability Type
Wrong Event Emission

Severity
Low

Description
sellBNB() emits SellToken(pair, BNB_ADDRESS, tokenOut, amountIn, actualAmountOut). The actual amount of native ETH spent is computed as actualAmountIn = beforeBalance - afterBalance, which can differ from amountIn (e.g., if the pair refunds ETH). Emitting amountIn misleads indexers and off-chain risk engines.

Affected Code

- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/liquidator/Liquidator.sol#L212

Impacts
Off-chain monitoring and accounting can mis-estimate slippage/execution quality, potentially masking anomalies or triggering false alerts.

Remediation
It is recommended to emit actualAmountIn instead of amountIn:- emit SellToken(pair, BNB_ADDRESS, tokenOut, actualAmountIn, actualAmountOut).

Retest
This issue has been fixed by emitting actualAmountIn in event.

Bug ID #L020 [Fixed]

Missing Price Feed Validation

Vulnerability Type
Lack of Input Validation (SC04-Lack Of Input Validation)

Severity
Low

Description
Chainlink has a library AggregatorV3Interface with a function called latestRoundData(). This function returns the price feed among other details for the latest round.
The contract was found to be using latestRoundData() without proper input validations on the returned parameters which might result in a stale and outdated price.

Affected Code

- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/ResilientOracle.sol#L228C3-L243C4

Impacts
Having oracles with functions to fetch price feed without any validation might introduce erroneous or invalid price values that could result in an invalid price calculation further in the contract.

Remediation
It is recommended to have input validations for all the parameters obtained from the Chainlink price feed. Here's a sample implementation:

```
(uint80 roundID ,int256 price, uint256 timestamp, uint80 answeredInRound) = AggregatorV3Interface(chainLinkAggregatorMap[underlying]).latestRoundData();

require(answer > 0, "Chainlink price <= 0");
require(answeredInRound >= roundID, "Stale price");
require(timestamp != 0, "Round not complete");
```

Retest
This issue has been fixed by introducing input validations.

Bug ID #I001 [Not Fixed]

Documentation Can Misrepresent Market Creation Permission Model For Users

Vulnerability Type
Poor Governance Documentation (SCWE-015)

Severity
Informational

Description
The protocol documentation explains that a Market represents an isolated lending pool defined by immutable parameters and emphasizes that market creation is intended to be permissionless, allowing any user to deploy a new market without governance approval. This description establishes the expected usage model and informs user trust assumptions about decentralization and openness of the system. However, the onchain implementation of createMarket enforces an access control requirement by checking whether the OPERATOR role exists and, if so, restricting market creation to accounts holding that role via hasRole(OPERATOR, msg.sender). This introduces a conditional permissioned flow that contradicts the documented claim of unconditional permissionless market creation.

Affected Code
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis/Ardentis.sol#L263

Impacts
Users, integrators, and auditors may incorrectly assume that new markets can always be created without centralized oversight, leading to misplaced trust in decentralization guarantees, inaccurate risk assessments, and potential reputational damage if the permissioned behavior is discovered post-deployment.

Remediation
It is recommended to update the documentation to accurately describe the conditional permissioning of market creation.

Retest
This issue has not been fixed.

Bug ID #I002 [Fixed]

Fee Transfers Can Be Failed Due To Rigid Ether Transfer Method

Vulnerability Type
Missing Best Practices

Severity
Informational

Description
The transferFee function is responsible for sending accrued protocol fees from the contract to a designated fee recipient and is restricted to an administrator or the vault owner. The function resets the recorded fee balance and then transfers Ether to the recipient using Solidity's transfer method, which forwards a fixed 2,300 gas stipend. This implementation assumes that the recipient can always receive Ether within this strict gas limit. However, if the recipient is a smart contract with a fallback or receive function requiring more gas, the transfer will revert. The root cause is reliance on transfer, which is no longer considered a robust Ether transfer mechanism due to evolving gas costs and execution requirements.

Affected Code
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/vault-allocator/VaultAllocator.sol#L109

Impacts
Fee withdrawals may unexpectedly revert, preventing authorized parties from claiming accrued fees and potentially causing operational disruptions or stuck protocol revenue.

Remediation
It is recommended to replace the use of transfer with a low-level call pattern and explicitly handle the success condition.

Retest
This issue has been fixed.

Bug ID #I003 [Fixed]

Protocol Cannot Recover Stuck Native Or Wrapped Tokens From Providers

Vulnerability Type
Missing functionality (SCWE-006)

Severity
Informational

Description
The ETHProvider and BNBProvider contracts act as routing layers that wrap and unwrap native assets into their wrapped equivalents and facilitate interactions with Ardentis vaults and the core Ardentis protocol. Under normal execution, native assets and wrapped tokens are expected to be fully consumed within a single transaction flow or returned to users. However, both contracts can still accumulate unintended balances due to forced native transfers, partial execution failures, unexpected external behavior, or future upgrades that alter control flow. Neither provider includes a privileged rescue or sweep function to recover accidentally stuck native assets or ERC20-compatible tokens. The root cause is the absence of an administrative recovery mechanism despite both contracts being upgradeable and role-controlled.

Affected Code
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/provider/BNBProvider.sol#L23
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/provider/ETHProvider.sol#L23

Impacts
If native assets or wrapped tokens become stuck in either provider contract, those funds may be permanently inaccessible, leading to avoidable asset loss.

Remediation
It is recommended to add a restricted rescue function to both provider contracts that allows an authorized role to withdraw arbitrary native assets or ERC20-compatible tokens.

Retest
This issue has been fixed.

Bug ID #I004 [Acknowledged]

Hardcoded Static Address

Vulnerability Type
Hardcoded Constants (SCWE-008)

Severity
Informational

Description
The contract was found to be using hardcoded addresses.
This could have been optimized using dynamic address update techniques along with proper access control to aid in address upgrade at a later stage.

Affected Code

- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis-vault/ArdentisVaultFactory.sol#L24
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/OracleAdaptor.sol#L12
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/OracleAdaptor.sol#L14
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/OracleAdaptor.sol#L16
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/OracleAdaptor.sol#L18
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/liquidator/Liquidator.sol#L34
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/liquidator/PublicLiquidator.sol#L38
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis-vault/ArdentisVaultFactory.sol#L24
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis-vault/ArdentisVaultFactoryInternal.sol#L22

Impacts
Hardcoding address variables in the contract make it difficult for it to be modified at a later stage in the contract as everything will need to be deployed again at a different address if there's a code upgrade.

Remediation

It is recommended to create dynamic functions to address upgrades so that it becomes easier for developers to make changes at a later stage if necessary.
The said function should have proper access controls to make sure only administrators can call that function using access control modifiers.
There should also be a zero address validation in the function to make sure the tokens are not lost.
If the address is supposed to be hardcoded, it is advisable to make it a constant if its value is not getting updated.

Retest
Client's Comment: Acknowledged.

Bug ID #I005 [Not Fixed]

Missing zero address validations

Vulnerability Type
Missing Input Validation (SC04-Lack Of Input Validation)

Severity
Informational

Description:
The contracts were found to be setting new addresses without proper validations for zero addresses.
Address type parameters should include a zero-address check otherwise contract functionality may become inaccessible or tokens burned forever.
Depending on the logic of the contract, this could prove fatal and the users or the contracts could lose their funds, or the ownership of the contract could be lost forever.

Affected Code

- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/ResilientOracle.sol#L107
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/vault-allocator/VaultAllocator.sol#L76C2-L80C4
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/OracleAdaptor.sol#L87
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/ResilientOracle.sol#L105
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/liquidator/Liquidator.sol#L117
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis-vault/ArdentisVault.sol#L451
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/provider/ETHProvider.sol#L291
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/provider/BNBProvider.sol#L325

Impacts
If address type parameters do not include a zero-address check, contract functionality may become unavailable or tokens may be burned permanently.

Remediation
Add a zero address validation to all the functions where addresses are being set.

Retest
This issue has not been fixed.

Bug ID #I006 [Not Fixed]

Documentation Can Misrepresent Supported Asset Tokens To Integrators

Vulnerability Type
Documentation Mismatch

Severity
Informational

Description
The ArdentisVaultFactory is designed to deploy new vaults with a single underlying asset, and the createArdentisVaultfunction enforces that this asset must use 18 decimals via IERC20Metadata(asset).decimals() == 18. This restriction ensures compatibility with the vault implementation and its internal accounting assumptions. However, the documented user flow and examples reference USDT as a supported asset token. USDT commonly uses 6 decimals, which does not satisfy the enforced requirement in the factory. As a result, the documentation presents an asset option that cannot be successfully used with the deployed contract, creating a discrepancy between expected and actual behavior.

Affected Code
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis-vault/ArdentisVaultFactory.sol#L76

Impacts
Integrators or partners may attempt to deploy vaults using USDT based on the documentation, only to encounter unexpected transaction reverts, leading to wasted integration effort, deployment delays, and reduced trust in the protocol's developer experience.

Remediation
It is recommended to align the documentation with the implemented constraint by explicitly stating that only 18-decimal tokens are supported or update the factory and vault logic to safely support non-18-decimal assets.

Retest
This issue has not been fixed as evidenced by the following link: [Link].

Bug ID #I007 [Not Fixed]

Price Calculation Assumes Presence of Optional decimals() Function

Vulnerability Type
Configuration Risk

Severity
Informational

Description

The getPrice() function unconditionally calls decimals() on both the collateral and loan tokens. According to the ERC-20 specification, decimals() is optional and not guaranteed to be implemented by all ERC20 tokens. If a token without decimals() is used, the call will revert and break price calculation and all dependent functionality.

Since market creation in Ardentis is permissionless, a market can be created using a token that does not implement decimals(), resulting in a permanently unusable market.

Affected Code

- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis/Ardentis.sol#L728-L737

Impacts
Markets using ERC20 tokens without decimals() will revert during price calculation, preventing borrowing, liquidation, and health checks.

Remediation
It is recommended to document that only ERC20 tokens implementing decimals() are supported, or add validation during market creation to ensure the token implements the decimals() function.

Retest
This issue has not been fixed.

Bug ID #I008 [Acknowledged]

Use Ownable2Step

Vulnerability Type
Missing Best Practices

Severity
Informational

Description
The "Ownable2Step" pattern is an improvement over the traditional "Ownable" pattern, designed to enhance the security of ownership transfer functionality in a smart contract. Unlike the original "Ownable" pattern, where ownership can be transferred directly to a specified address, the "Ownable2Step" pattern introduces an additional step in the ownership transfer process. Ownership transfer only completes when the proposed new owner explicitly accepts the ownership, mitigating the risk of accidental or unintended ownership transfers to mistyped addresses.

Affected Code
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/BoundValidator.sol#L15
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/ResilientOracle.sol#L36

Impacts
Without the "Ownable2Step" pattern, the contract owner might inadvertently transfer ownership to an unintended or mistyped address, potentially leading to a loss of control over the contract. By adopting the "Ownable2Step" pattern, the smart contract becomes more resilient against external attacks aimed at seizing ownership or manipulating the contract's behavior.

Remediation
It is recommended to use either Ownable2Step or Ownable2StepUpgradeable depending on the smart contract.

Retest:
Client's Comment: Acknowledged.

Bug ID #I009 [Fixed]

Lack of Access Control in reallocateTo Function

Vulnerability Type
Access Control (SCWE-016)

Severity
Informational

Description
The reallocateTo function enables rebalancing of a vault's supplied assets by withdrawing liquidity from one or more markets and reallocating it into another enabled market. This mechanism is intended to operate within predefined safety constraints, including flow caps, enabled market checks, and mandatory fee payment, ensuring reallocations remain bounded. However, the function is fully permissionless and does not enforce any access control or role-based authorization on the caller. As a result, any external user can trigger reallocations that are valid within configured limits but economically suboptimal for vault depositors. The root cause is the absence of caller authorization checks despite the function directly influencing vault allocation and reward distribution.

Affected Code

- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/vault-allocator/VaultAllocator.sol#L116

Impacts
External users can deliberately trigger reallocations that reduce vault yield or alter intended market exposure, potentially degrading depositor returns, even though protocol invariants and caps remain respected.

Remediation
It is recommended to restrict reallocateTo to authorized actors such as the vault owner, a designated manager role, or governance, or clearly document the intended permissionless design and associated economic trade-offs.

Retest
This issue has been fixed by adding access control in reallocateTo.

Bug ID #I010 [Not Fixed]

Oracle Decimal Mismatch Can Break Price Calculation For Markets

Vulnerability Type
Lack of Input Validation (SC04-Lack Of Input Validation)

Severity
Informational

Description
The getPrice function is used to compute the relative price between a collateral asset and a loan asset for an Ardentis market. To do so, it first queries the decimals of both tokens, then retrieves their respective oracle prices, and finally applies a scaling factor to normalize and calculate the relative price. This logic implicitly assumes that both oracle price feeds return values using the same number of decimals. However, while many common oracle providers return prices with uniform decimals, some valid oracle feeds use differing decimal configurations. The root cause is the lack of validation or normalization for oracle price decimals despite market creation being permissionless.

Affected Code
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis/Ardentis.sol#L728C2-L737C4

Impacts
Markets configured with oracle feeds that return prices using different decimals may compute incorrect prices, leading to broken market behavior and potential loss of funds for users interacting with those markets.

Remediation
It is recommended to explicitly document and enforce the requirement that both oracle price feeds used in a market must return prices with the same decimals, either through validation during market creation or clear documentation in code and external docs.

Retest
This issue has not been fixed.

Bug ID #G001 [Acknowledged]

Gas Optimization in Require/Revert Statements

Vulnerability Type
Gas Optimization (SCWE-082)

Severity
Gas

Description
The **require/revert** statement takes an input string to show errors if the validation fails.
The strings inside these functions that are longer than **32 bytes** require at least one additional MSTORE, along with additional overhead for computing memory offset and other parameters. For this purpose, having strings lesser than 32 bytes saves a significant amount of gas.

Affected Code

- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/timelock/TimeLock.sol#L16
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/revenue/LendingFeeRecipient.sol#L54
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/revenue/LendingFeeRecipient.sol#L55
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/revenue/LendingFeeRecipient.sol#L71
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/revenue/LendingFeeRecipient.sol#L72
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/revenue/LendingFeeRecipient.sol#L81
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/revenue/LendingFeeRecipient.sol#L82
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/BoundValidator.sol#L74
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/PTLinearDiscountOracle.sol#L40
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/PTLinearDiscountOracle.sol#L68
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/PTLinearDiscountOracle.sol#L91

- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/PTLinearDiscountOracle.sol#L58
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/PTLinearDiscountOracle.sol#L64
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/PTLinearDiscountMarketOracle.sol#L53
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/PTLinearDiscountMarketOracle.sol#L55
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/PTLinearDiscountMarketOracle.sol#L57
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/PTLinearDiscountMarketOracle.sol#L97
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/PTLinearDiscountMarketOracle.sol#L120
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/PTLinearDiscountMarketOracle.sol#L87
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/PTLinearDiscountMarketOracle.sol#L93
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/ResilientOracle.sol#L144
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/OracleAdaptor.sol#L37
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/OracleAdaptor.sol#L89
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/liquidator/Liquidator.sol#L429
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/liquidator/PublicLiquidator.sol#L78
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/liquidator/PublicLiquidator.sol#L90
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/liquidator/PublicLiquidator.sol#L445

Impacts
Having longer require/revert strings than **32 bytes** cost a significant amount of gas.

Remediation
It is recommended to shorten the strings passed inside **require/revert** statements to fit under **32 bytes**. This will decrease the gas usage at the time of deployment and at runtime when the validation condition is met.

Retest

Client's Comment: Acknowledged.

Bug ID #G002 [Not Fixed]

Cheaper conditional operators

Vulnerability Type
Gas Optimization (SCWE-082)

Severity
Gas

Description
Upon reviewing the code, it has been observed that the contract uses conditional statements involving comparisons with unsigned integer variables. Specifically, the contract employs the conditional operators x != 0 and x > 0 interchangeably. However, it's important to note that during compilation, x != 0 is generally more cost-effective than x > 0 for unsigned integers within conditional statements.

Affected Code

- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/vault-allocator/VaultAllocator.sol#L122
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/revenue/LendingRevenueDistributor.sol#L80
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/revenue/LendingRevenueDistributor.sol#L151
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/revenue/LendingRevenueDistributor.sol#L106
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/revenue/LendingRevenueDistributor.sol#L110
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/revenue/LendingRevenueDistributor.sol#L115
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis/Ardentis.sol#L303
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis/Ardentis.sol#L312
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis/Ardentis.sol#L338
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis/Ardentis.sol#L379

- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis/Ardentis.sol#L413
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis/Ardentis.sol#L423
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis/Ardentis.sol#L456
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis/Ardentis.sol#L577
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/PTLinearDiscountOracle.sol#L43
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/PTLinearDiscountOracle.sol#L58
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/PTLinearDiscountOracle.sol#L64
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/PTLinearDiscountMarketOracle.sol#L63
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/PTLinearDiscountMarketOracle.sol#L68
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/PTLinearDiscountMarketOracle.sol#L87
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/PTLinearDiscountMarketOracle.sol#L93
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/liquidator/Liquidator.sol#L160
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/liquidator/Liquidator.sol#L196
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/liquidator/Liquidator.sol#L344
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/liquidator/Liquidator.sol#L454
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/liquidator/Liquidator.sol#L461
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/liquidator/PublicLiquidator.sol#L362
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/liquidator/PublicLiquidator.sol#L371
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/liquidator/PublicLiquidator.sol#L401
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/liquidator/PublicLiquidator.sol#L471
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/liquidator/PublicLiquidator.sol#L478

- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/utils/BatchManagementUtils.sol#L54
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/interest-rate-model/FixedRateIrm.sol#L63
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/interest-rate-model/FixedRateIrm.sol#L77
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/interest-rate-model/InterestRateModel.sol#L226
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis-vault/ArdentisVault.sol#L659
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/provider/ETHProvider.sol#L75
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/provider/ETHProvider.sol#L90
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/provider/ETHProvider.sol#L116
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/provider/ETHProvider.sol#L142
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/provider/ETHProvider.sol#L209
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/provider/ETHProvider.sol#L239
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/provider/BNBProvider.sol#L111
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/provider/BNBProvider.sol#L125
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/provider/BNBProvider.sol#L151
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/provider/BNBProvider.sol#L177
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/provider/BNBProvider.sol#L243
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/provider/BNBProvider.sol#L273

Impacts

Employing x != 0 in conditional statements can result in reduced gas consumption compared to using x > 0. This optimization contributes to cost-effectiveness in contract interactions.

Remediation

Whenever possible, use the x != 0 conditional operator instead of x > 0 for unsigned integer variables in conditional statements.

Retest

This issue has not been fixed.

Bug ID #G003 [Not Fixed]

Custom error to save gas

Vulnerability Type
Gas Optimization (SCWE-082)

Severity
Gas

Description
During code analysis, it was observed that the smart contract is using the revert() statements for error handling. However, since Solidity version 0.8.4, custom errors have been introduced, providing a better alternative to the traditional revert(). Custom errors allow developers to pass dynamic data along with the revert, making error handling more informative and efficient. Furthermore, using custom errors can result in lower gas costs compared to the revert() statements.

Affected Code

- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/revenue/LendingFeeRecipient.sol#L113
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/BoundValidator.sol#L53
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/BoundValidator.sol#L72
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/BoundValidator.sol#L73
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/BoundValidator.sol#L74
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/BoundValidator.sol#L91
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/BoundValidator.sol#L92
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/PTLinearDiscountOracle.sol#L68
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/PTLinearDiscountOracle.sol#L91

- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/PTLinearDiscountMarketOracle.sol#L97
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/PTLinearDiscountMarketOracle.sol#L120
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/ResilientOracle.sol#L117
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/ResilientOracle.sol#L144
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/ResilientOracle.sol#L286

Impacts
Custom errors allow developers to provide more descriptive error messages with dynamic data. This provides better insights into the cause of the error, making it easier for users and developers to understand and address issues.

Remediation
It is recommended to replace all the instances of revert() statements with error() to save gas..

Retest
This issue has not been fixed.

Bug ID #G004 [Acknowledged]

Gas Optimization in Increments

Vulnerability Type
Gas optimization (SCWE-082)

Severity
Gas

Description
The contract uses two for loops**,** which use post increments for the variable "**i**".
The contract can save some gas by changing this to **++i**.
++i costs less gas compared to **i++** or **i += 1** for unsigned integers. In **i++**, the compiler has to create a temporary variable to store the initial value. This is not the case with **++i** in which the value is directly incremented and returned, thus, making it a cheaper alternative.

Vulnerable Code

- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/vault-allocator/VaultAllocator.sol#L91
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/vault-allocator/VaultAllocator.sol#L134
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/revenue/LendingFeeRecipient.sol#L93
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/revenue/LendingFeeRecipient.sol#L104
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/revenue/LendingFeeRecipient.sol#L119
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/revenue/LendingFeeRecipient.sol#L138
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/revenue/LendingFeeRecipient.sol#L150
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/revenue/LendingFeeRecipient.sol#L163
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/revenue/LendingRevenueDistributor.sol#L82
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/revenue/LendingRevenueDistributor.sol#L153

- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/ardentis/Ardentis.sol#L752
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/oracle/OracleAdaptor.sol#L38
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/liquidator/Liquidator.sol#L108
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/liquidator/Liquidator.sol#L118
- https://github.com/ardentis-lending/ardentis-contracts/blob/a6181f2e0e7edb0f94cbc240c080baa54f2847c9/src/utils/BatchManagementUtils.sol#L56

Impacts
Using **i++** instead of **++i** costs the contract deployment around 600 more gas units.

Remediation
It is recommended to switch to **++i** and change the code accordingly so the function logic remains the same and meanwhile saves some gas.

Retest
Client's Comment: Acknowledged.

6. The Disclosure ----------

The Reports provided by CredShields are not an endorsement or condemnation of any specific project or team and do not guarantee the security of any specific project. The contents of this report are not intended to be used to make decisions about buying or selling tokens, products, services, or any other assets and should not be interpreted as such.

Emerging technologies such as Smart Contracts and Solidity carry a high level of technical risk and uncertainty. CredShields does not provide any warranty or representation about the quality of code, the business model or the proprietors of any such business model, or the legal compliance of any business. The report is not intended to be used as investment advice and should not be relied upon as such.

CredShields Audit team is not responsible for any decisions or actions taken by any third party based on the report.

Your Secure Future Starts Here

At CredShields, we're more than just auditors. We're your strategic partner in ensuring a secure Web3 future. Our commitment to your success extends beyond the report, offering ongoing support and guidance to protect your digital assets

20A TANJONG PAGAR ROAD, SINGAPORE (088443)

info@credshields.com