QuillAudits

AUDIT REPORT

May, 2025

For

OIKOS

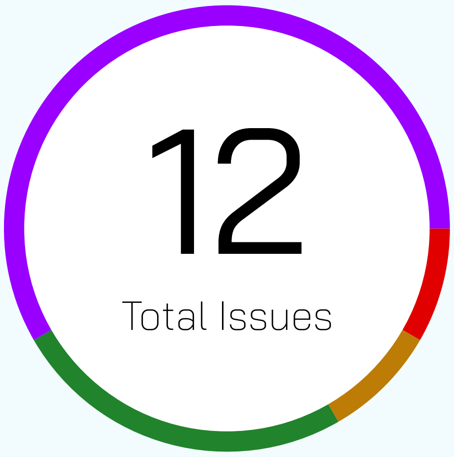
Table of Content

Executive Summary

Project name	Oikos
Project URL	https://oikos.cash/
Overview	Oikos Migration contract will allow withdrawals from a list of preloaded address based on IMV progression.
Audit Scope	The objective of this audit is to evaluate the security, code quality, and correctness of the Oikos codebase. The Oikos codebase is a fork of the Noma Protocol with the addition of a Migrator contract. The QuillAudits team focused primarily on auditing the Migrator contract to ensure its safety. We also verified that the remaining Oikos contracts are same to the audited Noma contracts, with the only differences being variable name changes—for example, replacing noma with oikos.
Source Code	https://github.com/oikos-cash/core/blob/audit_ready/src/bootstrap/Migration.sol
Contract in Scope	1. Primary focus: Okios Migration Contract. Path: src/bootstrap/Migration.sol 2. Differential Audit for oikos Codebase with respect to Audited Noma Codebase.
Branch	audit_ready
Commit Hash	76a687d5b6242eca73167c2fd5fdb6a0bddae9ae
Language	Solidity
Blockchain	BNB

Method	Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives.
Review 1	22nd April 2025 - 23rd April 2025
Updated code Received	20th May 2025
Review 2	20th May 2025 - 23rd May 2025
Fixed In	a5e990dd14e034029a598f917af79a0087ba0a56

Number of Issues per Severity



🟥	High	1 (8.33%)
🟧	Medium	1 (8.33%)
🟩	Low	3 (25.00%)
🟪	Informational	7 (58.33%)

Severity

Issues

	🟥 High	🟧 Medium	🟩 Low	🟪 Informational
Open	0	0	0	0
Resolved	**1**	**1**	**3**	0
Acknowledged	0	0	0	**7**
Partially Resolved	0	0	0	0

Checked Vulnerabilities

- ✓ Access Management
- ✓ Arbitrary write to storage
- ✓ Centralization of control
- ✓ Ether theft
- ✓ Improper or missing events
- ✓ Logical issues and flaws
- ✓ Arithmetic Computations Correctness
- ✓ Race conditions/front running
- ✓ SWC Registry
- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ Exception Disorder
- ✓ Gasless Send
- ✓ Use of tx.origin
- ✓ Malicious libraries
- ✓ Compiler version not fixed
- ✓ Address hardcoded
- ✓ Divide before multiply
- ✓ Integer overflow/underflow
- ✓ ERC's conformance
- ✓ Dangerous strict equalities
- ✓ Tautology or contradiction
- ✓ Return values of low-level calls
- ✓ Missing Zero Address Validation
- ✓ Private modifier
- ✓ Revert/require functions
- ✓ Multiple Sends
- ✓ Using suicide
- ✓ Using delegatecall
- ✓ Upgradeable safety
- ✓ Using throw

- ✓ Using inline assembly
- ✓ Unsafe type inference
- ✓ Style guide violation
- ✓ Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments, match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools And Platforms Used For Audit

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity statistical analysis.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below

🟥 High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

🟧 Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

🟩 Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

🟪 Informational Issues

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open Security vulnerabilities identified that must be resolved and are currently unresolved.	**Resolved** Security vulnerabilities identified that must be resolved and are currently unresolved.
Acknowledged Vulnerabilities which have been acknowledged but are yet to be resolved.	**Partially Resolved** Considerable efforts have been invested to reduce the risk/ impact of the security issue, but are not completely resolved.

High Severity Issues

withdraw() function transfers available balance instead of minAmountOut

Resolved

Path

src/bootstrap/Presale.sol

Function

withdraw()

Description

The withdraw() transfers the availableBalance for the token, instead it should be the amount calculated according to the number of pNoma tokens the user had. In withdraw() first it gets the balance of the user, then burns the p-assets. Then calculates the minAmountOut value. Finally it transfers the availableBalance instead of minAmountOut resulting in transfer of more than intended tokens.

Recommendation

To resolve the issue please make sure to change the availableBalance variable to minAmountOut in withdraw() function.

Fixed in

https://github.com/noma-protocol/core_contracts/commit/07738cfaaf56801bcde10e8bf12e790ca881efc3

Medium Severity Issues

Self-Referral Abuse Allows Users to Earn Discounts

Resolved

Path

src/bootstrap/Presale.sol

Function

deposit() & payReferrals()

Description

The deposit() function in the presale contract allows users to pass an arbitrary referralCode. However, this referralCode is later interpreted as the address to receive referral rewards in payReferrals()

- A user can generate their own referral code from their own address (e.g., bytes32(uint256(uint160(msg.sender)))) and pass it to the deposit() function.

- As a result, they receive a percentage of their own deposit back via the referral mechanism.

- This effectively gives them a discount on token purchases at the cost of the protocol.

Additionally:

- msg.value == 0 is allowed, which could result in unintended free mints or bloated contributor lists

Recommendation

- Restrict Self-Referrals:
* Check that
msg.sender != address(uint160(uint256(referralCode))) during deposit()

- Use a Mapping for Referral Attribution

- Best to use nonreentrant modifier and access control for the payReferrals function

Fixed in

8b04984163c4894185e1e1ca96bd3328d42f8186

Low Severity Issues

Presale can exceed hardcap

Resolved

Path

src/bootstrap/Presale.sol

Function

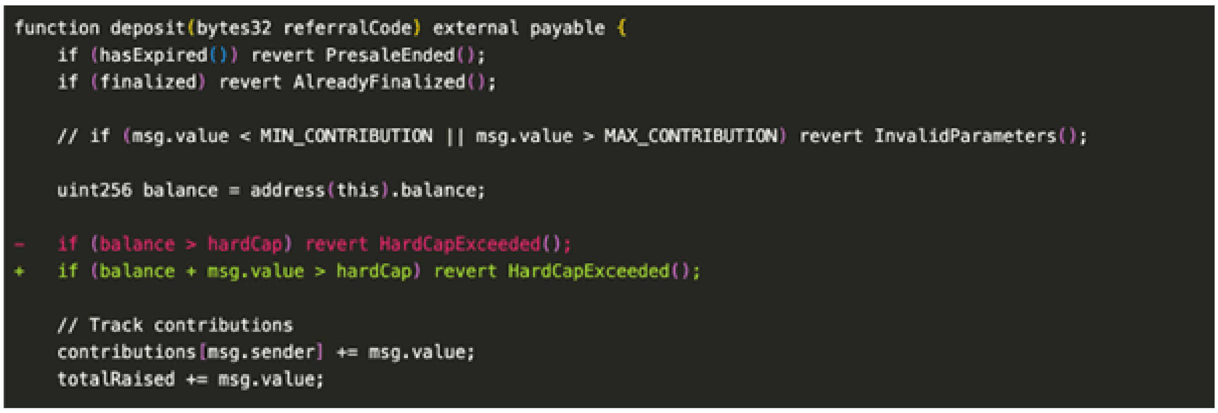
finalize()

Description

The deposit function in the Presale contract performs an incorrect validation check that could allow contributions to exceed the intended hardcap limit. The check is performed before the new contribution is added to the total, and it only reverts if the current balance is already greater than the hardcap.

Recommendation

Adjust the function this way to protect from overflowing the hardcap



```
function deposit(bytes32 referralCode) external payable {
    if (hasExpired()) revert PresaleEnded();
    if (finalized) revert AlreadyFinalized();

    // if (msg.value < MIN_CONTRIBUTION || msg.value > MAX_CONTRIBUTION) revert InvalidParameters();

    uint256 balance = address(this).balance;

-   if (balance > hardCap) revert HardCapExceeded();
+   if (balance + msg.value > hardCap) revert HardCapExceeded();

    // Track contributions
    contributions[msg.sender] += msg.value;
    totalRaised += msg.value;
```

Fixed in

06762639187916b5a9d171856b62f5d8fc723aef

Missing Emergency Switch, Contribution Constraints, and State Sync in Emergency Withdrawals

Resolved

Path

src/bootstrap/Presale.sol

Function

emergencyWithdrawal() & deposit()

Description

1. No Emergency Mode Toggle:
The emergencyWithdrawal() function does not depend on any emergency toggle/switch, meaning it's passively time-gated (deadline + 30 days), rather than actively controlled by the protocol or owner.

This limits response flexibility in critical failures (e.g., token price manipulation, stuck funds, or an attack on the protocol).

2. Improper Error Message:
The revert message in emergencyWithdrawal() uses PresaleOngoing() when finalized is true, which is semantically incorrect. If the presale is finalized, emergency withdrawal shouldn't be allowed at all, or a different error like PresaleFinalized() should be used.

3. State Inconsistency in Emergency Withdrawal:
While a user's individual contributions[msg.sender] is zeroed, the global totalRaised variable is not decremented.

This leads to inflated accounting, which can break invariant assumptions during finalization or analytics.

4. Commented-Out Contribution Limits:
In deposit(), the MIN_CONTRIBUTION and MAX_CONTRIBUTION checks are commented out

Recommendation

1. Introduce an Emergency Mode Switch

2. Replace PresaleOngoing() in emergencyWithdrawal() with a clearer message

3. Decrement totalRaised in the emergencyWithdrawal function

4. Remove comment and enforce min/max contribution limits

5. Add msg.value != 0 Check in deposit()

Fixed in

e0e1a1a11efc8ab0d5d1bfa696cf91eefca2a098

And

b40b773f28dc2c4b2d74ebcfddcb38100ad72a6a

And

5ce77cc86ff185c13a01878260d46e3200cd63ae

Comments mismatch with slippage value

Resolved

Path

src/bootstrap/Presale.sol

Function

finalize()

Description

In Presale contract, slippage check is calculated in the finalize() function to ensure the stability of the price. But the issue is that the comment has mentioned slippage to be 0.1% and the actual slippage calculated in contract is 0.5% which is higher and is mismatched from the sentence.

Recommendation

It is recommended to change slippage value to match the comment or comment value to match the calculated value

Fixed in

5dfa744e9e25b14f01c755bb3ef09291ba39dfb0

Informational Severity Issues

Ambiguity in onlyVault modifier allows both vault and staking contract to call protected functions

Acknowledged

Path

src/staking/Staking.sol

Description

The onlyVault modifier in the staking contract is currently ambiguous, allowing both the vault and staking contract to call functions protected by this modifier. This could lead to confusion about which contract is intended to have exclusive access to these functions.

Recommendation

Clarify the onlyVault modifier to ensure it aligns with the intended access control policy. If the intention is to allow only the vault contract to call these functions, update the modifier.

Noma's Team Comment

This is a design choice

Unused modifiers

Acknowledged

Path

src/vault/LendingVault.sol

Description

There are few modifiers which are not used in contracts.
- onlyVault in LendingVault

The codebase contains several unused functions, variables, and libraries that contribute to unnecessary complexity and potential confusion for developers and auditors. These unused elements can obscure the code's true functionality and make maintenance more challenging.

- MockNomaTokenRestricted.sol and Bootstrap.sol are unused.
- In StakingVault.sol, the LiquidityOps library and functions _collectLiquidity and _transferExcessBalance are defined but not used.
- In NomaFactory.sol, the teamMultiSig address is declared but never set, rendering it ineffective.
- In Presale.sol, the variables MIN_CONTRIBUTION and MAX_CONTRIBUTION are calculated in the constructor but never used.

Recommendation

To resolve the issue please remove the unused modifier, functions, variables, libraries or use them accordingly in the contract.

Redundant condition in TokenFactory's _deployNomaToken function

Acknowledged

Path

src/factory/TokenFactory.sol

Description

The _deployNomaToken function in TokenFactory.sol contains a redundant condition that checks if the proxy address is greater than or equal to _token1. This check is unnecessary because the logic of the do...while loop already ensures that this condition will never be true when the loop exits.

Recommendation

The check below can be removed because it is redundant.

```
if (address(proxy) >= _token1) revert InvalidTokenAddressError(); // Redundant check
```

Noma's Team comment :

This is necessary to force the order of tokens in the Uniswap V3 pair.

Mismatched NatSpec comments and code structures can cause confusion

Acknowledged

Path

src/factory/TokenFactory.sol

Description

The codebase contains NatSpec comments that do not accurately reflect the corresponding code structures. This mismatch can lead to confusion for devs trying to understand the intended functionality and usage of these structures.

Recommendation

Update the codebase NatSpec comments.

Identical Function Signatures: mintTokens(address,uint256) in Multiple Contracts

Acknowledged

Path

src/factory/NomaFactory.sol & src/vault/LendingVault.sol

Function

mintTokens()

Description

Both LendingVault::mintTokens() and NomaFactory::mintTokens() share the same function selector f0dda65c. This could lead to confusion in logs, tooling, or proxy setups, especially in cases where introspection or low-level calls are involved.

Recommendation

Consider renaming one of the functions to avoid selector collision

Noma's Team Comment

Actually, since NomaFactory isn't behind the Diamond proxy, there's no selector collision possible.

Missing Lender Controls Over Loan Rollovers

Acknowledged

Path

src/vault/LendingVault.sol

Function

rollLoan()

Description

The rollLoan function allows borrowers to extend (roll) their loan duration and increase borrow amounts, without any lender-side constraints or approval. There is no restriction on the number of rollovers, total duration, or borrower eligibility.

As a result:

- Borrowers may infinitely roll their loans
- Lenders may be locked into illiquid positions
- There is no mechanism for lenders to enforce repayment or liquidation, despite borrower expiry or undercollateralization risks

Recommendation

Introduce lender-side constraints and controls, such as:

- Maximum number of rollovers
- Maximum cumulative duration per loan

Noma's Team Comment

There is no undercollateralization risk as the loan can’t go under water.

Missing or Extra Struct Variables Compared to Doc-strings

Acknowledged

Path

src/types/Types.sol

Function

N/A

Description

Multiple struct definitions in the codebase contain discrepancies between their declared fields and the comments/documentation provided. This creates confusion and increases the likelihood of bugs, misconfigurations, or developer misunderstanding.

1. Struct: RewardParams
a. Missing: imv, spotPrice, totalSupply, kr
b. Unexpected: totalStaked (not mentioned in comment)

2. Struct: ProtocolParameters
a. Docstring omits: shiftAnchorUpperBips, slideAnchorUpperBips, and all the fee-related fields
b. These may be valid additions, but comments should be updated accordingly

3. Struct: LiquidityInternalPars
a. Docstring doesn't match the struct at all — likely refers to a different struct entirely
b. The struct itself is clear, but needs correct documentation

Recommendation

Synchronize docstrings with actual struct fields — ensure the comments reflect reality

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the Oikos Migration contract and differential audit for Oikos Codebase with respect to the audited Noma codebase. We performed our audit according to the procedure described above.

High , Medium , Low and informational severity issues were found . Oikos's team resolved few and acknowledged others

Disclaimer

At QuillAudits, we have spent years helping projects strengthen their smart contract security. However, security is not a one-time event—threats evolve, and so do attack vectors. Our audit provides a security assessment based on the best industry practices at the time of review, identifying known vulnerabilities in the received smart contract source code.

This report does not serve as a security guarantee, investment advice, or an endorsement of any platform. It reflects our findings based on the provided code at the time of analysis and may no longer be relevant after any modifications. The presence of an audit does not imply that the contract is free of vulnerabilities or fully secure.

While we have conducted a thorough review, security is an ongoing process. We strongly recommend multiple independent audits, continuous monitoring, and a public bug bounty program to enhance resilience against emerging threats.

Stay proactive. Stay secure.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem

QuillAudits

7+ Years of Expertise	**1M+** Lines of Code Audited
$30B+ Secured in Digital Assets	**1400+** Projects Secured

Follow Our Journey

AUDIT REPORT

May, 2025

For

OIKOS

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com