VIDMA

Nestcoin

SMART CONTRACT AUDIT

Project: Nestcoin
Date: May 26th, 2022

TABLE OF CONTENTS

SUMMARY

Vidma is pleased to present this audit report outlining our assessment of code, smart contracts, and other important audit insights and suggestions for management, developers, and users.

The audited contract is a new platform that provides easy exchange between fiat and crypto for customers design with unlimited possibilities. The codebase has been coded conforming to the EIP-2535 Diamond standard and is well documented and defined.

While the audit process no critical or high issues were detected. Audited smart contracts are written according to security best practices.

During the audit process, the Vidma team found several issues, including those with critical severity. A detailed summary and the current state are displayed in the table below.

Severity of the issue	Total found	Resolved	Unresolved
Critical	0 issues	0 issues	0 issues
High	0 issues	0 issues	0 issues
Medium	1 issue	1 issue	0 issues
Low	3 issues	3 issues	0 issues
Informational	1 issue	1 issue	0 issues
Total	**5 issues**	**5 issues**	**0 issues**

After evaluating the findings in this report and the final state after fixes, the Vidma auditors can state that the contracts are fully operational and secure. Under the given circumstances, we set the following risk level:

Our auditors are evaluating the initial commit given for the scope of the audit and the last commit with the fixes. This approach helps us adequately and sequentially evaluate the quality of the code. Code style, optimization of the contracts, the number of issues, and risk level of the issues are all taken into consideration. The Vidma team has developed a transparent scoring system presented below.

Severity of the issue	Resolved	Unresolved
Critical	1	10
High	0.8	7
Medium	0.5	5
Low	0.2	0.5
Informational	0	0.1

Please note that the points are deducted out of 100 for each and every issue on the list of findings (according to the current status of the issue). Issues marked as "not valid" are not subject to point deduction.

Based on the **overall result of the audit**, the Vidma audit team grants the following score:

In addition to manual check and static analysis, the auditing team has conducted a number of integrated autotests to ensure the given codebase has an adequate performance and security level.

The test results and the coverage can be found in the accompanying section of this audit report.

Please be aware that this audit does not certify the definitive reliability and security level of the contract. This document describes all vulnerabilities, typos, performance issues, and security issues found by the Vidma audit team. If the code is still under development, we highly recommend running one more audit once the code is finalized.

SCOPE OF WORK

nestcoin

They build, operate and invest in simple products that make crypto accessible to everyone.

Within the scope of this audit, two independent auditors thoroughly investigated the given codebase and analyzed the overall security and performance of the smart contracts.

The audit was conducted from May 5th to May 26th, 2022. The outcome is disclosed in this document.

The scope of work for the given audit consists of the following contracts:

- EscrowAccount;
- FactoryProxy;
- LookupProxy;
- LPoolAccount;
- EscrowAccountOps;
- EscrowOfframp;
- EscrowOnramp;
- FactoryOps;
- LookupOps;
- LPoolAccountOps;
- OwnableOps;
- ProxyAdmin.

The source code was taken from the following **source**:
https://github.com/NestcoinCo/onboard-onchain-escrow/tree

Initial commit submitted for the audit:
94f3ca66410f56375d99779514fc56382d02575e

Last commit reviewed by the auditing team:
3cdbcae31bde659b868dd2601493ab81d28e2236

As a reference to the contracts logic, business concept, and the expected behavior of the codebase, the Nestcoin team has provided the following documentation:
https://drive.google.com/drive/folders/1JZQ94XM_1BGqAUjcyXZ6euDQyY3eWga_?usp=sharing

WORKFLOW OF THE AUDITING PROCESS

Vidma audit team uses the most sophisticated and contemporary methods and well-developed techniques to ensure contracts are free of vulnerabilities and security risks. The overall workflow consists of the following phases:

Phase 1: The research phase

Research

After the Audit kick-off, our security team conducts research on the contract's logic and expected behavior of the audited contract.

Documentation reading

Vidma auditors do a deep dive into your tech documentation with the aim of discovering all the behavior patterns of your codebase and analyzing the potential audit and testing scenarios.

The outcome

At this point, the Vidma auditors are ready to kick off the process. We set the auditing strategies and methods and are prepared to conduct the first audit part.

Phase 2: Manual part of the audit

Manual check

During the manual phase of the audit, the Vidma team manually looks through the code in order to find any security issues, typos, or discrepancies with the logic of the contract. The initial commit as stated in the agreement is taken into consideration.

Static analysis check

Static analysis tools are used to find any other vulnerabilities in smart contracts that were missed after a manual check.

The outcome

An interim report with the list of issues.

Phase 3: Testing part of the audit

Integration tests

Within the testing part, Vidma auditors run integration tests using the Truffle or Hardhat testing framework. The test coverage and the test results are inserted in the accompanying section of this audit report.

The outcome

Second interim report with the list of new issues found during the testing part of the audit process.

STRUCTURE AND ORGANIZATION OF THE FINDINGS

For simplicity in reviewing the findings in this report, Vidma auditors classify the findings in accordance with the severity level of the issues. (from most critical to least critical).

All issues are marked as “Resolved” or “Unresolved”, depending on if they have been fixed by Nestcoin or not. The issues with “Not Valid” status are left on the list of findings but are not eligible for the score points deduction.

The latest commit with the fixes reviewed by the auditors is indicated in the “Scope of Work” section of the report.

The Vidma team always provides a detailed description of the issues and recommendations on how to fix them.

Classification of found issues is graded according to 6 levels of severity described below:

Critical

The issue affects the contract in such a way that funds may be lost or allocated incorrectly, or the issue could result in a significant loss.

Example: Underflow/overflow, precisions, locked funds.

High

The issue significantly affects the ability of the contract to compile or operate. These are potential security or operational issues.

Example: Compilation errors, pausing/unpausing of some functionality, a random value, recursion, the logic that can use all gas from block (too many iterations in the loop), no limitations for locking period, cooldown, arithmetic errors which can cause underflow, etc.

Medium

The issue slightly impacts the contract's ability to operate by slightly hindering its intended behavior.

Example: Absence of emergency withdrawal of funds, using assert for parameter sanitization.

Low

The issue doesn't contain operational or security risks, but are more related to optimization of the codebase.

Example: Unused variables, inappropriate function visibility (public instead of external), useless importing of SCs, misuse or disuse of constant and immutable, absent indexing of parameters in events, absent events to track important state changes, absence of getters for important variables, usage of string as a key instead of a hash, etc.

Informational

Are classified as every point that increases onboarding time and code reading, as well as the issues which have no impact on the contract's ability to operate.

Example: Code style, NatSpec, typos, license, refactoring, naming convention (or unclear naming), layout order, functions order, lack of any type of documentation.

MANUAL REPORT

Missing zero address validation at EscrowOnramp.sol

Medium | Resolved

In function *approveOnrampOrder()* (lines 23, 33) there is no check for zero address of *_client*, so it's possible to create an order for zero address. Additionally, in functions *cancelOnrampOrder()* and *confirmOnrampOrder()* there is a check for zero address client which can lead to issues with canceling created by mistake order.

Recommendation:

Add check for zero address in function *_beforeOnramp()*.

Local variables shadowing

Low | Resolved

FactoryOps.sol in external functions *getPoolAccounts()* (line 56), *getPoolAccountAddress()* (line 64), *createEscrowAccount()* (line 74), *createPoolAccount()* (line 94) and internal function implementations *_owner* variable is used which shadowing existing *_owner* from Ownable.sol

Recommendation:

Rename the local variables that shadow another component.

Re-Audit:

Only part of recommendation was applied. Function *createPoolAccount()* (line 93) and internal function implementations *_createEscrowAccount()*, *_createLPoolAccount()* wasn't changed.

Not standard DiamondCut event

Low | Resolved

According to EIP-2535 any function or code that adds or replaces or removes one or more functions MUST emit the standard DiamondCut event:

```
event DiamondCut(FacetCut[] _diamondCut, address _init, bytes
_calldata);
```

It's needed for proper records of all changes to a diamond.

Recommendation:

Change event according to EIP-2535.

Missing zero address validation at EscrowOfframp.sol

Low | Resolved

In function *_createOfframpOrder()* (line 180) there is no check for zero address of the *_client*.

Recommendation:

Add check for zero address in function *_createOfframpOrder()*.

Unexpected error return when there is no pool account for *_poolOwner*

Informational | Resolved

Function *getPoolAccountAddress()* return address of pool account for pool owner at a specific index. When the pool owner has no valid pool account should return a zero address but a revert occurs instead in line 66 *count -1*. In this case, count equals 0.

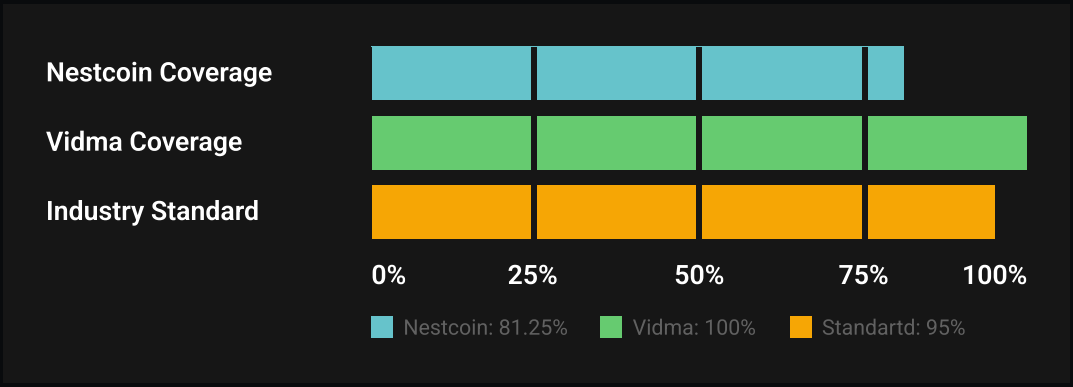
Recommendation:

Consider handling a case when the count of pool accounts equals 0 by providing an additional check.

TEST RESULTS

To verify the security of contracts and the performance, a number of integration tests were carried out using the Truffle testing framework.

In this section, we provide both tests written by Nestcoin and tests written by Vidma auditors.



It is important to note that Vidma auditors do not modify, edit or add tests to the existing tests provided in the Nestcoin repository. We write totally separate tests with code coverage of a minimum of 95% to meet the industry standards.

Tests written by Nestcoin

Test Coverage

File	% Stmts	% Branch	% Funcs	% Lines
contracts\	100.00	100.00	100.00	100.00
EscrowAccount.sol	100.00	100.00	100.00	100.00
FactoryProxy.sol	100.00	100.00	100.00	100.00
LPoolAccount.sol	100.00	100.00	100.00	100.00
LookupProxy.sol	100.00	100.00	100.00	100.00
contracts\implementation\	86.93	58.04	76.24	86.85
EscrowAccountOps.sol	53.85	100.00	30.77	53.85
EscrowOfframp.sol	95.92	56.82	93.1	94.95
EscrowOnramp.sol	92.45	53.57	84.62	92.45
FactoryOps.sol	83.33	62.5	66.67	84.44
LPoolAccountOps.sol	93.02	65.00	100.00	93.33
LookupOps.sol	68.42	58.33	55.56	68.42
OwnableOps.sol	50.00	100.00	50.00	50.00
contracts\upgrade\	50.00	25.00	50.00	50.00

File	% Stmts	% Branch	% Funcs	% Lines
ProxyAdmin.sol	50.00	25.00	50.00	50.00
All Files	**81.25**	**81.5**	**83.3**	**81.25**

Test Results

```
Onboard Escrow Factory
  ✓ Should deploy proxy and set implementation (271ms)
  ✓ Should setup proxy At implementation (101ms)
  ✓ Should reject only owner allowed call (58ms)
  ✓ Should transfer ownership (74ms)
  ✓ Should allow TP to create escrow account (191ms)
  ✓ Should revert intermediary account creation from unathorized
    account with:
NOT_AUTHORIZED
  ✓ Should increment tp accounts count (174ms)
  ✓ Should revert with: ACCOUNT_LIMIT_REACHED (40ms)
Lookup and Escrow Ops
  ✓ Should revert with: NO_IMPLEMENTATION on unregsitered
    implementation (53ms)
  ✓ Should revert revert diamondCut call with: NOT_OWNER
  ✓ Should allow diamond cut from Owner (243ms)
  ✓ Should get owner from ownable facet (64ms)
  ✓ Should returns balances (41ms)
  ✓ Should receive network coin (68ms)
  ✓ Should receive ERC20 token (86ms)
  ✓ Should revert with: NOT_OWNER for withdrawal
  ✓ Should revert with: NOT_OWNER for ERC20 withdrawal (73ms)
  ✓ Should send available balance (115ms)
  ✓ Should send available ERC20 balance (144ms)
Escrow Offramp
  ✓ Should deploy EscrowOfframp contract (94ms)
  ✓ Should register EscrowOfframp facets (228ms)
  ✓ Should create offramp order and lock amount (104ms)
  ✓ Should require approval for token offramp (61ms)
  ✓ Should create token offramp order (158ms)
```

```
    ✓ Should revert order cancel action from Cutomer with:
      NOT_AUTHORIZED
    ✓ Should revert ERC20 order cancel action from Customer with:
      NOT_AUTHORIZED
    ✓ Should allow TP to cancel order and refunds customer (66ms)
    ✓ Should allow TP to cancel ERC20 order and refunds customer (84ms)
    ✓ Should revert confirm payment action from TP with:
      NOT_MEDIATOR (79ms)
    ✓ Should allow mediator to confirm payment action and increase
      escrow available balance (102ms)
    ✓ Should allow customer to confirm order (161ms)
    ✓ Should revert ERC20 order confirm payment action from TP with:
      NOT_MEDIATOR (143ms)
    ✓ Should allow mediator to confirm ERC20 order payment action and
      available balance (116ms)
    ✓ Should allow customer to confirm ERC20 order (240ms)
  Escrow Onramp
    ✓ Should delpoy EscrowOnramp Contract (78ms)
    ✓ Should register Onramp signatures and revert with
      ZERO_AMOUNT (164ms)
    ✓ Should revert with INSUFFICIENT_ESCROW_BALANCE (96ms)
    ✓ Should revert with INVALID_TOKEN_ADDRESS (86ms)
    ✓ Should create On-Ramp Order and lock amount (112ms)
    ✓ Should create On-Ramp order ERC20 and lock balance (121ms)
    ✓ Should revert cancel order from TP with: NOT_MEDIATOR
    ✓ Should allow customer to cancel order and released locked
      asset (87ms)
    ✓ Should allow mediator to cancel ERC20 order and release
      locked asset (168ms)
    ✓ Should revert confirm order from customer with:
      NOT_AUTHORIZED (250ms)
    ✓ Should confirm order by TP and send asset to customer (86ms)
    ✓ Should confirm order by Mediator and send asset to
      customer (81ms)
  Implementation Upgrade
    ✓ Should upgrade factory immplementation (171ms)
    ✓ Should upgrade lookp implementation (139ms)
  Offramp From Liquidity Pool Account
    ✓ Should deploy LPoolAccountOps contract (58ms)
    ✓ Should register LPoolAccountOps to Lookup and lock
      implementation (200ms)
    ✓ Should create exchange liquidity pool account (131ms)
```

```
  ✓ Should allow call from registered global implemetation (50ms)
  ✓ Should deposit native coin to pool account (82ms)
  ✓ Should deposit ERC20 token to pool account (55ms)
  ✓ Should revert offramp with NOT_OWNER
  ✓ hould not offramp more than available balance:
    reverts with LOW_POOL_BALANCE
  ✓ Should offramp native coin available in pool balance and lock
    it in escrow (143ms)
  ✓ Should revert offramp with repeated `orderId` with
    ORDER_ID_EXISTS
  ✓ Should offramp ERC20 token available in pool balance and lock
    it in escrow (186ms)
  ✓ Should get matching order order details
  ✓ Should allow offramp of native coin with valid signature (181ms)
  ✓ Should allow offramp of ERC20 token with valid signature (210ms)
  ✓ Should prevent offramp confirmation if not from owner or
    mediator (48ms)
  ✓ Should confirm offramp order and increase escrow available
    balance (113ms)
  ✓ Should revert confirmation of completed order with
    INVALID_ORDER (63ms)
  ✓ Should confirm offramp order and increase escrow available
    token balance using mediator account (143ms)
  ✓ Should cancel offramp order and refund pool account (172ms)
  ✓ Should revert withdrawal call from non mediator or owner with
    NOT_AUTHORIZED (64ms)
  ✓ Should revert withdrawal above pool account balance with
    LOW_TOKEN_BALANCE (70ms)
  ✓ Should withdrawal available to owner address (99ms)
  ✓ Should not register external escrow in factory (125ms)
  ✓ Should revert with ESCROW_NOT_REGISTERED for external
    escrow account (93ms)
  ✓ Should revert with NOT_POOL_ACCOUNT for external liquidity
    pool account (171ms)

74 passing (13s)
```

Tests written by Vidma auditors

Test Coverage

File	% Stmts	% Branch	% Funcs	% Lines
contracts\	100.00	100.00	100.00	100.00
EscrowAccount.sol	100.00	100.00	100.00	100.00
FactoryProxy.sol	100.00	100.00	100.00	100.00
LPoolAccount.sol	100.00	100.00	100.00	100.00
LookupProxy.sol	100.00	100.00	100.00	100.00
contracts\implementation\	100.00	93.75	100.00	100.00
EscrowAccountOps.sol	100.00	80.00	100.00	100.00
EscrowOfframp.sol	100.00	78.94	100.00	100.00
EscrowOnramp.sol	100.00	95.45	100.00	100.00
FactoryOps.sol	100.00	100.00	100.00	100.00
LPoolAccountOps.sol	100.00	96.88	100.00	100.00
LookupOps.sol	100.00	95.00	100.00	100.00
OwnableOps.sol	100.00	100.00	100.00	100.00
contracts\upgrade\	100.00	90.00	100.00	100.00

File	% Stmts	% Branch	% Funcs	% Lines
ProxyAdmin.sol	100.00	90.00	100.00	100.00
All Files	**100.00**	**95.9**	**100.00**	**100.00**

Test Results

```
Contract: EscrowAccountOps
  ✓ should return details of new created escrow account correct
    correct (207ms)
  ✓ should set supported interfaces correct (233ms)
  ✓ should set mediator correct (141ms)
  ✓ should set autoEscrowAllowed correct (91ms)
  ✓ should return balance of native network coin correct
    [no locked amount] (314ms)
  ✓ should return balance of ERC20 token correct
    [no locked amount] (445ms)
  ✓ should withdraw native network coin correct
    [no locked amount] (1336ms)
  ✓ should withdraw ERC20 token correct [no locked amount] (1542ms)
  ✓ shouldn't withdraw [native and erc20 coins] by not
    the owner (921ms)
  ✓ shouldn't withdraw too big amount [native and erc20
    coins] (839ms)
  ✓ should setAutoEscrow by the owner correct (454ms)
  ✓ shouldn't setAutoEscrow by not the owner (328ms)
  ✓ should getEscrowStatus correct [no orders were made] (618ms)

Contract: EscrowOfframp
  createOfframpOrder [native network coin]
    ✓ should createOfframpOrder correct (786ms)
    ✓ shouldn't createOfframpOrder if client address is owner
      or mediator (780ms)
    ✓ shouldn't createOfframpOrder with zero amount (350ms)
    ✓ shouldn't createOfframpOrder if client has pending
      orders (1105ms)
```

```
    ✓ shouldn't createOfframpOrder if passed amount is less than
      real send coins amount (428ms)
    ✓ shouldn't withdraw escrow account if there is pending orders
      and all balance is locked (1128ms)
    ✓ should emit event (490ms)
  createOfframpOrder [ERC20 token]
    ✓ should createOfframpOrder correct (698ms)
    ✓ shouldn't createOfframpOrder if client address is owner
      or mediator (545ms)
    ✓ shouldn't createOfframpOrder with zero amount (457ms)
    ✓ shouldn't createOfframpOrder when token address
      is invalid (435ms)
    ✓ shouldn't createOfframpOrder if client has pending
      orders (1326ms)
    ✓ shouldn't withdraw escrow account if there is pending orders
      and all balance is locked (1169ms)
    ✓ should emit event (467ms)
  createOfframpOrder by mediator
    ✓ should createOfframpOrder correct (782ms)
    ✓ shouldn't createOfframpOrder if client address is owner
      or mediator (689ms)
    ✓ shouldn't createOfframpOrder by not mediator (430ms)
    ✓ shouldn't createOfframpOrder if client is zero address (418ms)
    ✓ shouldn't createOfframpOrder if client and token is the same
      address (416ms)
  createOfframpOrder by pool
    ✓ shouldn't createOfframpOrder if client address is owner
      or mediator (657ms)
    ✓ shouldn't createOfframpOrder if caller is not registered
      pool account (656ms)
    ✓ shouldn't createOfframpOrder if orederId is incorrect (676ms)
  cancelOfframpOrder by client [native network coin]
    ✓ should cancelOfframpOrder and refund client correct (1357ms)
    ✓ shouldn't cancelOfframpOrder if there is no pending
      orders for client (264ms)
    ✓ shouldn't cancelOfframpOrder if client is zero address (344ms)
    ✓ shouldn't cancelOfframpOrder by no the mediator (308ms)
  cancelOfframpOrder by client [ERC20 token]
    ✓ should cancelOfframpOrder and refund client correct (2141ms)
    ✓ shouldn't cancelOfframpOrder if there is no pending
      orders for client (311ms)
```

```
    ✓ shouldn't cancelOfframpOrder if token address is
      invalid (374ms)
    ✓ shouldn't cancelOfframpOrder if client is zero address (343ms)
    ✓ shouldn't cancelOfframpOrder by not the owner or
      the mediator (326ms)
  cancelOfframpOrder by owner or mediator to LPoolAccount
  [native network coin]
    ✓ should cancelOfframpOrder and refund client correct (1474ms)
    ✓ shouldn't cancelOfframpOrder by not the owner
      or mediator (326ms)
  cancelOfframpOrder by owner or mediator to LPoolAccount
  [ERC20 token]
    ✓ should cancelOfframpOrder and refund client correct (1820ms)
    ✓ shouldn't cancelOfframpOrder by not the owner or
      the mediator (313ms)
  confirmOfframpOrder by client [native network coin]
    ✓ should confirmOfframpOrder and approveOnrampOrder
      after correct (2103ms)
    ✓ shouldn't confirmOfframpOrder if there is no pending
      order (370ms)
    ✓ should withdraw escrow balance correct after confirm (2113ms)
  confirmOfframpOrder by client [ERC20 token]
    ✓ should confirmOfframpOrder correct (1596ms)
    ✓ shouldn't confirmOfframpOrder if there is no pending
      order (404ms)
    ✓ shouldn't confirmOfframpOrder if token address is
      invalid (296ms)
    ✓ should withdraw escrow balance correct after confirm (2393ms)
  confirmOfframpOrder by mediator [native network coin]
    ✓ should confirmOfframpOrder correct (1518ms)
    ✓ shouldn't confirmOfframpOrder if client address is zero (346ms)
    ✓ shouldn't confirmOfframpOrder by not the mediator (395ms)
  confirmOfframpOrder by mediator [ERC20 token]
    ✓ should confirmOfframpOrder correct (1737ms)
    ✓ shouldn't confirmOfframpOrder if client address is zero
    ✓ shouldn't confirmOfframpOrder if token address is
      invalid (327ms)
    ✓ shouldn't confirmOfframpOrder by not the mediator

Contract: EscrowOnramp
  approveOnrampOrder [native network coin]
    ✓ should approveOnrampOrder correct (749ms)
```

```
  ✓ shouldn't approveOnrampOrder by not the owner (466ms)
  ✓ should approveOnrampOrder by the mediator correct (1389ms)
  ✓ shouldn't approveOnrampOrder if client address is owner
    or mediator (696ms)
  ✓ shouldn't approveOnrampOrder if insufficient escrow
    balance (1372ms)
  ✓ shouldn't approveOnrampOrder zero amount (410ms)
  ✓ shouldn't approveOnrampOrder for zero address (464ms)
  ✓ shouldn't approveOnrampOrder if client has pending
    orders (923ms)
  ✓ shouldn't withdraw escrow account if there is pending orders
    and all balance is locked (981ms)
  ✓ should emit event (455ms)
approveOnrampOrder [ERC20 token]
  ✓ should approveOnrampOrder correct (1308ms)
  ✓ shouldn't approveOnrampOrder by not the owner (578ms)
  ✓ shouldn't approveOnrampOrder if token address is
    incorrect (313ms)
  ✓ hould approveOnrampOrder by the mediator correct (1588ms)
  ✓ shouldn't approveOnrampOrder if client address is owner
    or mediator (812ms)
  ✓ shouldn't approveOnrampOrder if insufficient escrow
    balance (1496ms)
  ✓ shouldn't approveOnrampOrder zero amount (621ms)
  ✓ shouldn't approveOnrampOrder for zero address (562ms)
  ✓ shouldn't approveOnrampOrder if client has pending
    orders (1123ms)
  ✓ shouldn't withdraw escrow account if there is pending orders
    and all balance is locked (1144ms)
  ✓ should emit event (576ms)
cancelOnrampOrder by client[native network coin]
  ✓ should cancelOnrampOrder and unlock balance correct (1643ms)
  ✓ shouldn't cancelOnrampOrder if there is no pending orders
    for msg.sender (359ms)
  ✓ should withdraw after order is canceled by the owner
    correct (2118ms)
cancelOnrampOrder by client [ERC20 token]
  ✓ should cancelOnrampOrder and unlock balance correct (1871ms)
  ✓ shouldn't cancelOnrampOrder if there is no pending orders
    for msg.sender (405ms)
  ✓ shouldn't cancelOnrampOrder if token address is
    incorrect (385ms)
```

```
    ✓ should withdraw after order is canceled by the owner
      correct (2152ms)
  cancelOnrampOrder by mediator[native network coin]
    ✓ should cancelOnrampOrder and unlock balance correct (1556ms)
    ✓ shouldn't cancelOnrampOrder if there is no pending orders
      for msg.sender (374ms)
    ✓ shouldn't cancelOnrampOrder by no the mediator (373ms)
    ✓ shouldn't cancelOnrampOrder if the client address is
      zero (309ms)
    ✓ should withdraw after order is canceled by the owner
      correct (2034ms)
  cancelOnrampOrder by mediator [ERC20 token]
    ✓ should cancelOnrampOrder and unlock balance correct (1654ms)
    ✓ shouldn't cancelOnrampOrder if there is no pending orders
      for msg.sender (327ms)
    ✓ shouldn't cancelOnrampOrder if token address is
      incorrect (354ms)
    ✓ shouldn't cancelOnrampOrder by no the mediator (312ms)
    ✓ shouldn't confirmOnrampOrder if the client address is
      zero (362ms)
    ✓ should withdraw after order is canceled by the owner
      correct (2452ms)
  confirmOnrampOrder [native network coin]
    ✓ should confirmOnrampOrder correct (1699ms)
    ✓ shouldn't cancelOnrampOrder if there is no pending orders
      for msg.sender (358ms)
    ✓ shouldn't confirmOnrampOrder if the client address is
      zero (328ms)
    ✓ shouldn't confirmOnrampOrder by not the owner or
      mediator (312ms)
  confirmOnrampOrder [ERC20 token]
    ✓ should confirmOnrampOrder correct (2092ms)
    ✓ shouldn't confirmOnrampOrder if there is no pending orders
      client address (355ms)
    ✓ shouldn't confirmOnrampOrder if the client address is
      zero (328ms)
    ✓ shouldn't confirmOnrampOrder by not the owner or
      mediator (297ms)

Contract: FactoryOps
  ✓ should create TP escrow account correct (582ms)
```

```
  ✓ shouldn't create new TP escrow account is reached
    userAccountLimit (1481ms)
  ✓ should create escrow account by owner correct | mediator
    onbehalf of TP (641ms)
  ✓ shouldn't create escrow account while passed TP address is
    zero (232ms)
  ✓ shouldn't create escrow account by not and owner or
    mediator (284ms)
  ✓ should create exchange pool account where message sender
    is owner (652ms)
  ✓ should create exchange pool account delegating as pool owner
    passed address (758ms)
  ✓ shouldn't create escrow account by not and owner or
    mediator (275ms)
  ✓ should check if passed address isEscrowAccount correct (661ms)
  ✓ should check if passed address isLPooolAccount correct (661ms)
  ✓ should setFactoryMediator correct (463ms)
  ✓ shouldn't create escrow account while passed TP address
    is zero (289ms)
  ✓ shouldn't create escrow account by not and owner or
    mediator (236ms)
  ✓ should get owner correct (111ms)
  ✓ should transfer ownership correct (289ms)

Contract: FactoryProxy
  should initialize correct
    ✓ should set owner correct (151ms)
    ✓ should set mediator correct (112ms)
    ✓ hould set proxy admin correct (124ms)
    ✓ should set implementation correct (109ms)
    ✓ should set escrow lookup correct (125ms)
    ✓ should set supported interfaces correct (214ms)

Contract: LookupOps
  ✓ should add diamondCut[IDiamondCut.FacetCut[]] correct (1767ms)
  ✓ should add diamondCut[IDiamondCut.FacetCut[], address ,bytes]
    correct (1775ms)
  ✓ shouldn't add new facet by not the owner (765ms)
  ✓ should lockImplementation to specific contract identifier
    correct (2802ms)
  ✓ shouldn't lock implementation by not the owner (235ms)
  ✓ should setGlobalUse for specific facet correct (2929ms)
```

```
    ✓ shouldn't set global use by not the owner (345ms)
    ✓ should get owner correct (110ms)
    ✓ should transfer ownership correct (411ms)

Contract: LPoolAccountOps
  balance of pool account
    ✓ should return balance of network native coin correct (325ms)
    ✓ should return balance of ERC20 token correct (387ms)
  offrampCoin
    ✓ should offrampCoin native coin from liquidy pool account
      correct (700ms)
    ✓ should offrampCoin native coin and confirmOfframpOrder
      correct (842ms)
    ✓ shouldn't offrampCoin by not the owner (280ms)
    ✓ should offrampCoin with signature correct (611ms)
    ✓ shouldn't offrampCoin with signature by not the
      mediator (446ms)
    ✓ shouldn't offrampCoin if pool balance is too low (312ms)
    ✓ shouldn't offrampCoin with the duplicated orederId (1143ms)
    ✓ shouldn't offrampCoin for unregistered escrow account (479ms)
  approveAndOfframp
    ✓ should approveAndOfframp correct (931ms)
    ✓ should approveAndOfframp and confirmOfframpOrder
      correct (1007ms)
    ✓ shouldn't approveAndOfframp by not the owner (545ms)
    ✓ should approveAndOfframp with signature correct (705ms)
    ✓ shouldn't approveAndOfframp with signature by not the
      mediator (734ms)
    ✓ shouldn't approveAndOfframp when orderId is duplicated (1203ms)
    ✓ shouldn't approveAndOfframp when erc20 token balance is
      too low (586ms)
    ✓ shouldn't approveAndOfframp for unregistered escrow (577ms)
  withdrawal [native coin and ERC20]
    ✓ should withdrawal native token correct (469ms)
    ✓ shouldn't withdrawal native token if balance is too
      small (495ms)
    ✓ should withdrawal ERC20 token correct (628ms)
    ✓ shouldn't withdrawal ERC20 token if balance is too
      small (560ms)
    ✓ should revert if incorrect token address (307ms)
    ✓ shouldn't withdraw by not the owner or mediator (594ms)
```

```
Contract: LookupProxy
  should initialize correct
    ✓ should set owner correct (110ms)
    ✓ should set proxy admin correct (140ms)
    ✓ should set implementation correct (94ms)
    ✓ should set supported interfaces correct (306ms)

Contract: OwnableOps
  ✓ should return owner correct (125ms)
  ✓ should transfer ownership correct (344ms)
  ✓ shouldn't transfer ownership by not the owner (338ms)
  ✓ should renounce ownership correct (351ms)
  ✓ shouldn't renounce ownership by not the owner (335ms)

Contract: ProxyAdmin
  ✓ should getProxyAdmin correct (139ms)
  ✓ should getProxyImplementation correct (150ms)
  ✓ should changeProxyAdmin correct (527ms)
  ✓ should upgrade correct (965ms)
  ✓ should upgradeAndCall correct (861ms)

177 passing (30m)
```

We are delighted to have a chance to work with the Nestcoin team and contribute to your company's success by reviewing and certifying the security of your smart contracts

The statements made in this document should be interpreted neither as investment or legal advice, nor should its authors be held accountable for decisions made based on this document.

Vidma is a security audit company helping crypto companies ensure their code and products operate safely and as intended, enabling founders to sleep soundly at night. We specialize in auditing DeFi protocols, layer one protocols, and marketplace solutions. Our team consists of experienced and internationally trained specialists. Our company is based in Ukraine, known for its strong engineering, cryptography, and cybersecurity culture.

Website: **vidma.io**
Email: **security@vidma.io**