CAIRO SECURITY CLAN

OPUS

SECURITY ASSESMENT REPORT

FEBRUARY 2025

Prepared for

LINDY LABS

Contents

1 About Cairo Security Clan

Cairo Security Clan is a leading force in the realm of blockchain security, dedicated to fortifying the foundations of the digital age. As pioneers in the field, we specialize in conducting meticulous smart contract security audits, ensuring the integrity and reliability of decentralized applications built on blockchain technology.

At Cairo Security Clan, we boast a multidisciplinary team of seasoned professionals proficient in blockchain security, cryptography, and software engineering. With a firm commitment to excellence, our experts delve into every aspect of the Web3 ecosystem, from foundational layer protocols to application-layer development. Our comprehensive suite of services encompasses smart contract audits, formal verification, and real-time monitoring, offering unparalleled protection against potential vulnerabilities.

Our team comprises industry veterans and scholars with extensive academic backgrounds and practical experience. Armed with advanced methodologies and cutting-edge tools, we scrutinize and analyze complex smart contracts with precision and rigor. Our track record speaks volumes, with a plethora of published research papers and citations, demonstrating our unwavering dedication to advancing the field of blockchain security.

At Cairo Security Clan, we prioritize collaboration and transparency, fostering meaningful partnerships with our clients. We believe in a customer-oriented approach, engaging stakeholders at every stage of the auditing process. By maintaining open lines of communication and soliciting client feedback, we ensure that our solutions are tailored to meet the unique needs and objectives of each project.

Beyond our core services, Cairo Security Clan is committed to driving innovation and shaping the future of blockchain technology. As active contributors to the ecosystem, we participate in the development of emerging technologies such as Starknet, leveraging our expertise to build robust infrastructure and tools. Through strategic guidance and support, we empower our partners to navigate the complexities of the blockchain landscape with confidence and clarity.

In summary, Cairo Security Clan stands at the forefront of blockchain security, blending technical prowess with a client-centric ethos to deliver unparalleled protection and peace of mind in an ever-evolving digital landscape. Join us in safeguarding the future of decentralized finance and digital assets with confidence and conviction.

2 Disclaimer

Disclaimer Limitations of this Audit:

This report is based solely on the materials and documentation provided by you to Cairo Security Clan for the specific purpose of conducting the security review outlined in the Summary of Audit and Scoped Files. The findings presented here may not be exhaustive and may not identify all potential vulnerabilities. Cairo Security Clan provides this review and report on an "as-is" and "as-available" basis. You acknowledge that your use of this report, including any associated services, products, protocols, platforms, content, and materials, occurs entirely at your own risk.

Inherent Risks of Blockchain Technology:

Blockchain technology remains in its developmental stage and is inherently susceptible to unknown risks and vulnerabilities. This review is specifically focused on the smart contract code and does not extend to the compiler layer, programming language elements beyond the reviewed code, or other potential security risks outside the code itself.

Report Purpose and Reliance:

This report should not be construed as an endorsement of any specific project or team, nor does it guarantee the absolute security of the audited smart contracts. No third party should rely on this report for any purpose, including making investment or purchasing decisions.

Liability Disclaimer:

To the fullest extent permitted by law, Cairo Security Clan disclaims all liability associated with this report, its contents, and any related services and products arising from your use. This includes, but is not limited to, implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Third-Party Products and Services:

Cairo Security Clan does not warrant, endorse, guarantee, or assume responsibility for any products or services advertised by third parties within this report, nor for any open-source or third-party software, code, libraries, materials, or information linked to, referenced by, or accessible through this report, its content, and related services and products. This includes any hyperlinked websites, websites or applications appearing on advertisements, and Cairo Security Clan will not be responsible for monitoring any transactions between you and third-party providers. It is recommended that you exercise due diligence and caution when considering any third-party products or services, just as you would with any purchase or service through any medium.

Disclaimer of Advice:

FOR THE AVOIDANCE OF DOUBT, THIS REPORT, ITS CONTENT, ACCESS, AND/OR USE, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHOULD NOT BE CONSIDERED OR RELIED UPON AS FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER PROFESSIONAL ADVICE.

3 Executive Summary

This document presents the security review performed by Cairo Security Clan on the Opus.

Opus is a cross-margin, autonomous Decentralized Digital Bank (DDB) designed to promote financial sovereignty by eliminating reliance on centralized power structures and traditional fiat systems.

Unlike conventional financial institutions that provide forward guidance only to make last-minute policy shifts, Opus operates with full autonomy. Its monetary policy is not determined by bureaucratic entities, centralized authorities, or even decentralized governance structures. Instead, it functions without human intervention, ensuring a transparent and predictable financial environment.

Opus enables users to borrow against a diversified portfolio of carefully selected collateral, including yield-bearing assets. Interest rates, maximum loan-to-value ratios, and liquidation thresholds are dynamically adjusted based on each user's collateral profile, minimizing the need for manual oversight.

The platform's first digital asset, CASH, is an algorithmic, overcollateralized stablecoin that is soft-pegged to the US dollar, offering a stable and secure medium of exchange within the Opus ecosystem. Learn more from docs.

The audit was performed using

- manual analysis of the codebase,
- automated analysis tools,
- simulation of the smart contract,
- analysis of edge test cases

5 points of attention, where 1 is classified as Critical, 0 is classified as High, 0 is classified as Medium,1 is classified as Low,0 is classified as Informational and 3 are classified as Best Practices. The issues are summarized in Fig. 1.

This document is organized as follows. Section 1 About Cairo Security Clan. Section 2 Disclaimer. Section 3 Executive Summary. Section 4 Summary of Audit. Section 5 Risk Classification. Section 6 Issues by Severity Levels. Section 7 Test Evaluation.

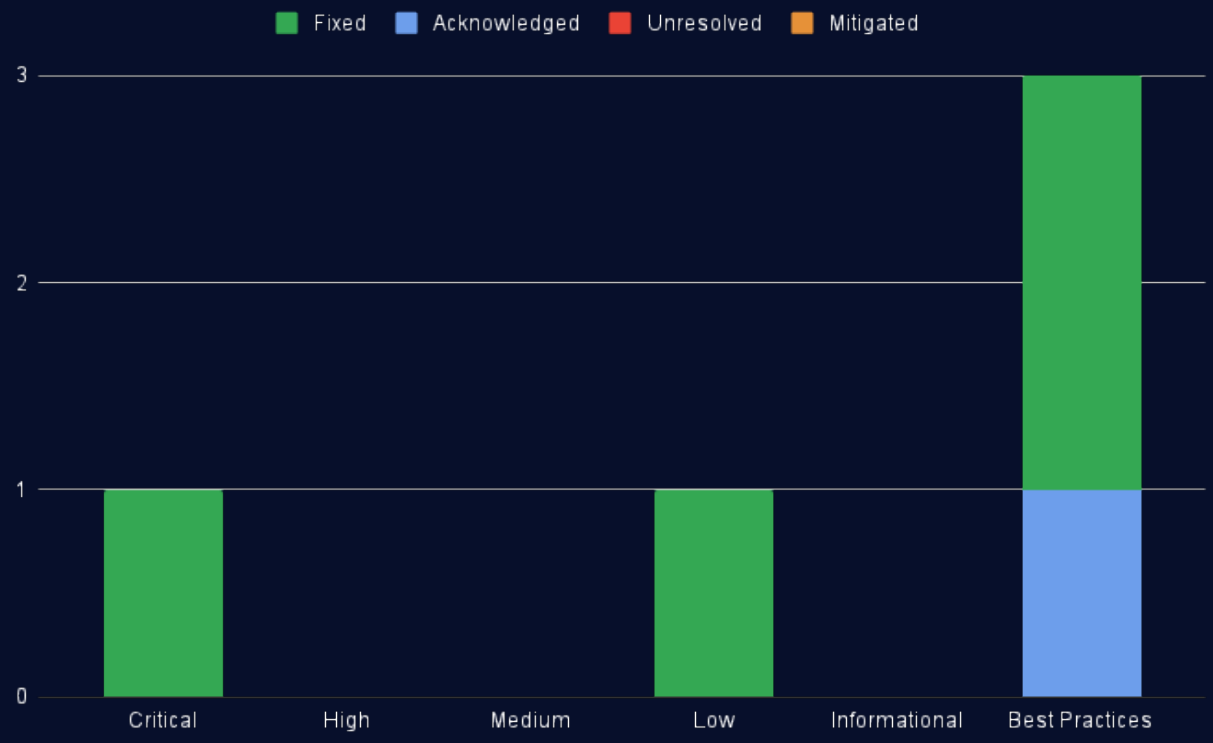

Fig 1: Distribution of issues: Critical (1), **High** (0), **Medium** (0), **Low** (1), **Informational** (0), **Best Practices** (3). **Distribution of status: Fixed** (4), **Acknowledged** (1), **Mitigated** (0), **Unresolved** (0).

4 Overview

Opus is an automated credit solution designed to run without human intervention, with common financial parameters such as interest rates, loan-to-value ratios, and liquidation thresholds being managed dynamically by the protocol. Users are able to open lines of credit by depositing collateral, which they can borrow against and repay their loan in the future with interest. It is possible to gain leveraged exposure to collateral assets, with the maximum leverage tailored to the user based on their maximum loan-to-value-ratio.

This audit engagement is primarily focused on the following contracts, some of which were newly introduced and others which are changes of existing contracts:

Our System Overview

- **Lever:** A newly introduced contract which allows users to increase or decrease their leverage on a given collateral asset in their trove.
- **Stabilizer:** A newly introduced contract where users can deposit Ekubo position NFTs and be rewarded with yin for the yield they generate. Users can stake and unstake at any time.
- **Receptor:** A newly introduced contract which determines the spot price of `yin` by selecting median oracle price between three stablecoin assets. The newly calculated price is passed to the `shrine` contract where it is updated.
- **Transmuter:** A new `v2` implementation of the transmuter that allows users to exchange collateral for `yin` when `yin` is greater than the peg price. It is also possible to reverse `yin` to collateral if the contract's settings have been configured to allow it.
- **Seer:** A contract with changes applied, that determines the current price of each `yang` asset by querying oracles, and updates the `yang` asset price on the `shrine` contract.

5 Summary of Audit

Audit Type	Security Review
Scarb Version of opus_contracts	2.6.5
Scarb Version of opus_compose	2.9.2
Final Report	14/02/2025
Repository of opus_contracts	lindy-labs/opus_contracts
Repository of opus_compose	lindy-labs/opus_compose
Initial Commit Hash of opus_compose	c3de72751f40ea3e1c4af283ffc8247b0fe98e8c
Documentation	Website documentation
Test Suite Assessment	High

5.1 Scoped Files

5.1.1 Fully Scoped Files

	Contracts
1	opus_contracts/src/core/transmuter_v2.cairo
2	opus_contracts/src/external/ekubo.cairo
3	opus_contracts/src/core/receptor.cairo
4	opus_contracts/src/utils/ekubo_oracle_adapter.cairo
5	opus_compose/src/stabilizer/contracts/stabilizer.cairo
6	opus_compose/src/stabilizer/interfaces/stabilizer.cairo
7	opus_compose/src/stabilizer/constants.cairo
8	opus_compose/src/stabilizer/math.cairo
9	opus_compose/src/stabilizer/types.cairo
10	opus_compose/src/lever/contracts/lever.cairo
11	opus_compose/src/lever/interfaces/lever.cairo
12	opus_compose/src/lever/constants.cairo
13	opus_compose/src/lever/types.cairo
14	opus_compose/src/interfaces/erc20.cairo

5.1.2 Partially Scoped Files

In `src/external/pragma_v2.cairo` and `src/core/seer_v2.cairo`, our scope is limited to the differences between commits.

In `src/utils/math.cairo`, the scope is restricted to the `convert_ekubo_oracle_price_to_wad` function.

	Contracts
1	opus_contracts/src/external/pragma_v2.cairo
2	opus_contracts/src/core/seer_v2.cairo
3	opus_contracts/src/utils/math.cairo

5.2 Issues

	Findings	Severity	Update
1	Direct call to `on_flash_loan` allows asset theft	Critical	Fixed
2	Lever up can forge a full debt amount while receiving less collateral	Low	Fixed
3	Duplicate emission of event `YieldStateUpdated`	Best Practices	Acknowledged
4	Re-use of existing role for other purpose	Best Practices	Fixed
5	Unused import	Best Practices	Fixed

6 Risk Classification

The risk rating methodology used by Cairo Security Clan follows the principles established by the CVSS risk rating methodology. The severity of each finding is determined by two factors: **Likelihood** and **Impact**.

Likelihood measures how likely an attacker will uncover and exploit the finding. This factor will be one of the following values:

a) **High**: The issue is trivial to exploit and has no specific conditions that need to be met;

b) **Medium**: The issue is moderately complex and may have some conditions that need to be met;

c) **Low**: The issue is very complex and requires very specific conditions to be met.

When defining the likelihood of a finding, other factors are also considered. These can include but are not limited to Motive, opportunity, exploit accessibility, ease of discovery, and ease of exploit.

Impact is a measure of the damage that may be caused if an attacker exploits the finding. This factor will be one of the following values:

a) **High**: The issue can cause significant damage such as loss of funds or the protocol entering an unrecoverable state;

b) **Medium**: The issue can cause moderate damage such as impacts that only affect a small group of users or only a particular part of the protocol;

c) **Low**: The issue can cause little to no damage such as bugs that are easily recoverable or cause unexpected interactions that cause minor inconveniences.

When defining the impact of a finding other factors are also considered. These can include but are not limited to Data/state integrity, loss of availability, financial loss, and reputation damage. After defining the likelihood and impact of an issue, the severity can be determined according to the table below.

		Likelihood		
		High	**Medium**	**Low**
Impact	**High**	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Info/Best Practices

To address issues that do not fit a High/Medium/Low severity, Cairo Security Clan also uses three more finding severities: **Informational**, **Best Practices** and **Gas**

a) **Informational** findings do not pose any risk to the application, but they carry some information that the audit team intends to formally pass to the client;

b) **Best Practice** findings are used when some piece of code does not conform with smart contract development best practices;

c) **Gas** findings are used when some piece of code uses more gas than it should be or have some functions that can be removed to save gas.

7 Issues by Severity Levels

7.1 Critical

7.1.1 Direct call to `on_flash_loan` allows asset theft

File(s): lever.cairo

Description: The `lever` contract allows users to adjust their leverage for a given trove ID through the functions `up` or `down`. The expected logic flow for both functions is:

1. A user calls either function.
2. The contract checks trove ownership.
3. The contract calls `flash_mint` with instructions to callback to `on_flash_loan`.
4. The leverage adjustment is completed using the flash-minted yin tokens.

The trove ownership check is performed within the `up` and `down` functions, as shown below:

```
fn up(ref self: ContractState, amount: Wad, lever_up_params: LeverUpParams) {
    let user: ContractAddress = get_caller_address();
    assert!(
        user == self.abbot.read()
            .get_trove_owner(lever_up_params.trove_id)
            .expect("Non-existent trove"),
        "LEV: Not trove owner",
    );
    // ...
}
```

Under normal protocol behavior, the function `on_flash_loan` should only be called by the `flash_mint` contract during the callback. However, there are no caller address checks present in the function, allowing any address to call it directly. This bypasses the trove ownership checks enforced in `up` and `down`.

An attacker can invoke `on_flash_loan` directly, controlling all arguments, including action data, without ownership validation. Using the `LeverDown` action, an attacker can manipulate arguments to convert any user's collateral into yin and transfer it to an arbitrary address. By setting the `amount` argument to zero, no yin will be used to repay the target trove's debt, allowing all withdrawn collateral to be swapped for yin and returned to an attacker-controlled address.

Example Attack Flow:

```
// Target trove data
let victim_trove_id = 555;

// Max extractable collateral
let steal_amount = 10_000000000000000000;

// Maliciously crafted leverdown params
let lever_down_params = LeverDownParams {
    trove_id: victim_trove_id,  // victim trove
    yang: yang_addr,            // collateral addr
    yang_amt: steal_amount,     // yin amount to steal
    swaps: steal_swap,          // swap data for N yang to steal_amount yin
}

// Steps that would execute inside `on_flash_loan`:
// melt(trove_id, 0)            // Zero debt burned
// exit(yang, steal_amount)     // Yang withdrawn
// withdraw()                   // Shrine accounting updated
// transfer()                   // Yang moved to swap router
// multi_multihop_swap()        // Stolen yang swapped to yin
// clear_minimum(0)             // Passes since zero
// transfer()                   // All yin is considered excess, sent to attacker
```

Impact: Troves with lower Loan-to-Value (LTV) ratios are more susceptible to losses since they hold more collateral that can be withdrawn before exceeding the LTV limit. The maximum amount that can be stolen can be estimated as:

```
// initial_yang: amount of collateral
// yang_price: price in yin
// current_debt: debt in yin
// threshold: LTV threshold
let max_yang = initial_yang - (( current_debt * 100 / ( threshold * yang_price ));
let expected_yin = max_yang * yang_price;
```

Recommendation(s): The function `on_flash_loan` should only allow calls from the `flash_mint` contract that were initiated by the `lever` contract itself. This can be enforced by:

- Adding a caller address check to ensure only `flash_mint` can invoke the function.
- Including an `initiator` argument check to validate that the call originated from `lever`'s own address.

Status: Fixed

Update from the client: Fixed in commit 4caec4a

7.2 Low

7.2.1 Lever up can forge a full debt amount while receiving less collateral

File(s): lever.cairo

Description: The `lever` function `up` allows users to leverage their positions by flash-minting yin, swapping it for collateral, and adding it to their trove. The swap from yin to collateral is executed using an Ekubo swap, with the specific swap parameters determined by Ekubo's API. However, there are no checks to ensure that the value of the collateral being deposited matches the value of the debt being created. This issue is highlighted in the following code snippet:

```
let LeverUpParams { trove_id, yang, max_forge_fee_pct, swaps } = params;

// Transfer yin and execute swap to collateral
yin.transfer(router.contract_address, amount);
router.multi_multihop_swap(swaps);

// Withdraw collateral, lower limit is only 1 wei
let asset_amt: u256 = router_clear.clear_minimum(yang_erc20, 1);

// Deposit purchased collateral to trove
yang_erc20.approve(gate, asset_amt);
let asset_amt: u128 = asset_amt.try_into().unwrap();
let yang_amt: Wad = sentinel.enter(yang, initiator, asset_amt);
shrine.deposit(yang, trove_id, yang_amt);

// Borrow `amount` yin from trove and send to this contract to repay mint
shrine.forge(initiator, trove_id, amount.try_into().unwrap(), max_forge_fee_pct);
```

There are no safeguards to ensure that the amount of collateral received from the Ekubo swap is within a reasonable range, as the function `clear_minimum(yang_erc20, 1)` allows any amount greater than 1 wei. Regardless of the collateral amount received, the amount of debt created on the target trove remains fixed at the `amount` argument.

If the yin value of the swapped collateral is lower than the expected debt amount, the function `on_flash_loan` will still create the full debt while the trove receives insufficient collateral value. This can lead to capital inefficiency, as users might forge more debt than what is actually utilized for leverage.

Impact: This mismatch between collateral and debt can result in positions with higher Loan-to-Value (LTV) ratios than intended. The greater the discrepancy between received collateral and the created debt, the higher the LTV. Elevated LTVs may hinder users' ability to manage their positions, potentially preventing collateral withdrawals or position closures—especially after interest accrual.

The severity of this issue is categorized as **Low**, as the likelihood of receiving mismatched or incorrect swap data is minimal. However, in the rare case that such a discrepancy occurs, it could impact the risk profile of users' positions.

Recommendation(s): Modify the check in `clear_minimum` to ensure that a reasonable amount of collateral assets is received relative to the new debt amount, instead of setting the minimum amount to 1.

Status: Fixed

Update from the client: Fixed in commit 23a5f48

7.3 Best Practices

7.3.1 Duplicate emission of event `YieldStateUpdated`

File(s): stabilizer.cairo

Description: The `YieldStateUpdated` event is designed to be emitted whenever the yield state of the stabilizer is updated. However, during the execution of the functions `unstake()` and `claim()`, this event is emitted twice within the same transaction.

This duplication occurs because both the `harvest()` and `withdraw_yield()` functions, which are invoked during these operations, independently trigger the `YieldStateUpdated` event.

Recommendation(s): Since `harvest()` only computes and returns the yield state without updating storage, consider removing the `YieldStateUpdated` event emission from this function.

Status: Acknowledged

Update from the client: The intermediate yield state in a harvest operation is useful for constructing the history of the contract's yin balance, before any yin is withdrawn to the caller as yield.

7.3.2 Re-use of existing role for other purpose

File(s): receptor.cairo

Description: Opus contracts implement finely tuned role management, ensuring that each privileged function is associated with a distinct role that directly reflects its behavior. However, in `receptor.cairo`, the function `set_oracle_extension` reuses the `SET_QUOTE_TOKENS` role, despite it not being relevant to the function's purpose.

Recommendation(s): Assign a role that accurately represents the function's behavior to maintain consistency and clarity in role-based access control.

Status: Fixed

Update from the client: Fixed in commit db826e7

7.3.3 Unused import

File(s): ekubo.cairo

Description: The file `ekubo.cairo` imports the function `convert_ekubo_oracle_price_to_wad` from `opus::utils::math`, but the function is never used. Unused code can reduce readability and should be removed where possible.

Recommendation(s): Remove the unused import to improve code clarity and maintainability.

Status: Fixed

Update from the client: Fixed in commit 10fcaa3

8 Test Evaluation

8.1 Compilation Output

```
scarb build
   Updating git repository github.com/ekuboprotocol/abis
Downloading access_control v0.4.0
Downloading opus v1.0.2
Downloading wadray v0.5.0
Downloading snforge_std v0.35.0
Downloading snforge_scarb_plugin v0.35.0
   Updating crates.io index
    Locking 123 packages to latest compatible versions
     Adding smol_str v0.2.2 (available: v0.3.2)
Downloading crates ...
 Downloaded serde_spanned v0.6.8
 Downloaded cairo-lang-debug v2.9.2
 Downloaded cairo-lang-macro-attributes v0.1.0
 Downloaded zerofrom v0.1.5
 Downloaded write16 v1.0.0
 Downloaded icu_provider_macros v1.5.0
 Downloaded relative-path v1.9.3
 Downloaded cairo-lang-macro v0.1.1
 Downloaded equivalent v1.0.1
 Downloaded scarb-stable-hash v1.0.0
 Downloaded cairo-lang-macro-stable v1.0.0
 Downloaded linkme-impl v0.3.31
 Downloaded cairo-lang-primitive-token v1.0.0
 Downloaded windows-targets v0.52.6
 Downloaded tinystr v0.7.6
 Downloaded yoke-derive v0.7.5
 Downloaded path-clean v1.0.1
 Downloaded form_urlencoded v1.2.1
 Downloaded writeable v0.5.5
 Downloaded xshell-macros v0.2.7
 Downloaded smol_str v0.2.2
 Downloaded utf8_iter v1.0.4
 Downloaded synstructure v0.13.1
 Downloaded xxhash-rust v0.8.15
 Downloaded utf16_iter v1.0.5
 Downloaded idna_adapter v1.2.0
 Downloaded stable_deref_trait v1.2.0
 Downloaded data-encoding v2.7.0
 Downloaded percent-encoding v2.3.1
 Downloaded genco-macros v0.17.10
 Downloaded lazy_static v1.5.0
 Downloaded cairo-lang-diagnostics v2.9.2
 Downloaded rand_core v0.6.4
 Downloaded indoc v2.0.5
 Downloaded version_check v0.9.5
 Downloaded num-integer v0.1.46
 Downloaded cfg-if v1.0.0
 Downloaded toml_datetime v0.6.8
 Downloaded thiserror v1.0.69
 Downloaded linkme v0.3.31
 Downloaded semver v1.0.25
 Downloaded heck v0.4.1
 Downloaded xshell v0.2.7
 Downloaded rust-analyzer-salsa-macros v0.17.0-pre.6
 Downloaded scopeguard v1.2.0
 Downloaded colored v2.2.0
 Downloaded rustc-hash v1.1.0
 Downloaded genco v0.17.10
 Downloaded icu_locid_transform v1.5.0
 Downloaded quote v1.0.38
 Downloaded cairo-lang-parser v2.9.2
 Downloaded lock_api v0.4.12
```

```
  Downloaded once_cell v1.20.3
  Downloaded litemap v0.7.4
  Downloaded displaydoc v0.2.5
  Downloaded pin-project-lite v0.2.16
  Downloaded zerovec-derive v0.10.3
  Downloaded unescaper v0.1.5
  Downloaded zerofrom-derive v0.1.5
  Downloaded smallvec v1.13.2
  Downloaded thiserror-impl v1.0.69
  Downloaded yoke v0.7.5
  Downloaded triomphe v0.1.14
  Downloaded cairo-lang-filesystem v2.9.2
  Downloaded parking_lot_core v0.9.10
  Downloaded icu_locid_transform_data v1.5.0
  Downloaded icu_locid v1.5.0
  Downloaded redox_syscall v0.5.8
  Downloaded toml v0.8.20
  Downloaded icu_normalizer v1.5.0
  Downloaded icu_provider v1.5.0
  Downloaded tracing-core v0.1.33
  Downloaded serde_derive v1.0.217
  Downloaded allocator-api2 v0.2.21
  Downloaded rand v0.8.5
  Downloaded indexmap v2.7.1
  Downloaded tracing v0.1.41
  Downloaded toml_edit v0.22.23
  Downloaded cairo-lang-syntax v2.9.2
  Downloaded hashbrown v0.15.2
  Downloaded zerocopy v0.7.35
  Downloaded winnow v0.7.1
  Downloaded hashbrown v0.14.5
  Downloaded aho-corasick v1.1.3
  Downloaded icu_properties_data v1.5.0
  Downloaded regex v1.11.1
  Downloaded idna v1.0.3
  Downloaded itertools v0.12.1
  Downloaded zerovec v0.10.4
  Downloaded syn v2.0.98
  Downloaded num-bigint v0.4.6
  Downloaded memchr v2.7.4
  Downloaded icu_collections v1.5.0
  Downloaded url v2.5.4
  Downloaded icu_properties v1.5.1
  Downloaded serde v1.0.217
  Downloaded bitflags v2.8.0
  Downloaded parking_lot v0.12.3
  Downloaded tracing-attributes v0.1.28
  Downloaded ahash v0.8.11
  Downloaded num-traits v0.2.19
  Downloaded icu_normalizer_data v1.5.0
  Downloaded proc-macro2 v1.0.93
  Downloaded unicode-ident v1.0.16
  Downloaded zerocopy-derive v0.7.35
  Downloaded either v1.13.0
  Downloaded oorandom v11.1.4
  Downloaded cairo-lang-syntax-codegen v2.9.2
  Downloaded cairo-lang-utils v2.9.2
  Downloaded autocfg v1.4.0
  Downloaded regex-syntax v0.8.5
  Downloaded windows_x86_64_gnullvm v0.52.6
  Downloaded windows_aarch64_gnullvm v0.52.6
  Downloaded windows-sys v0.59.0
  Downloaded windows_i686_gnullvm v0.52.6
  Downloaded rust-analyzer-salsa v0.17.0-pre.6
  Downloaded regex-automata v0.4.9
  Downloaded libc v0.2.169
  Downloaded windows_aarch64_msvc v0.52.6
  Downloaded windows_x86_64_gnu v0.52.6
  Downloaded windows_x86_64_msvc v0.52.6
  Downloaded windows_i686_gnu v0.52.6
```

```
  Downloaded windows_i686_msvc v0.52.6
   Compiling snforge_scarb_plugin v0.35.0
   Compiling proc-macro2 v1.0.93
   Compiling serde v1.0.217
   Compiling libc v0.2.169
   Compiling num-traits v0.2.19
   Compiling ahash v0.8.11
   Compiling lock_api v0.4.12
   Compiling parking_lot_core v0.9.10
   Compiling either v1.13.0
   Compiling scopeguard v1.2.0
   Compiling tracing-core v0.1.33
   Compiling pin-project-lite v0.2.16
   Compiling icu_locid_transform_data v1.5.0
   Compiling semver v1.0.25
   Compiling heck v0.4.1
   Compiling winnow v0.7.1
   Compiling itertools v0.12.1
   Compiling oorandom v11.1.4
   Compiling rustc-hash v1.1.0
   Compiling thiserror v1.0.69
   Compiling icu_properties_data v1.5.0
   Compiling genco-macros v0.17.10
   Compiling utf16_iter v1.0.5
   Compiling icu_normalizer_data v1.5.0
   Compiling write16 v1.0.0
   Compiling path-clean v1.0.1
   Compiling linkme-impl v0.3.31
   Compiling utf8_iter v1.0.4
   Compiling xxhash-rust v0.8.15
   Compiling relative-path v1.9.3
   Compiling data-encoding v2.7.0
   Compiling xshell-macros v0.2.7
   Compiling lazy_static v1.5.0
   Compiling percent-encoding v2.3.1
   Compiling quote v1.0.38
   Compiling cairo-lang-primitive-token v1.0.0
   Compiling form_urlencoded v1.2.1
   Compiling scarb-stable-hash v1.0.0
   Compiling syn v2.0.98
   Compiling colored v2.2.0
   Compiling cairo-lang-macro-stable v1.0.0
   Compiling indoc v2.0.5
   Compiling xshell v0.2.7
   Compiling num-integer v0.1.46
   Compiling parking_lot v0.12.3
   Compiling synstructure v0.13.1
   Compiling serde_derive v1.0.217
   Compiling zerofrom-derive v0.1.5
   Compiling yoke-derive v0.7.5
   Compiling zerovec-derive v0.10.3
   Compiling displaydoc v0.2.5
   Compiling icu_provider_macros v1.5.0
   Compiling tracing-attributes v0.1.28
   Compiling rust-analyzer-salsa-macros v0.17.0-pre.6
   Compiling thiserror-impl v1.0.69
   Compiling cairo-lang-macro-attributes v0.1.0
   Compiling genco v0.17.10
   Compiling linkme v0.3.31
   Compiling cairo-lang-macro v0.1.1
   Compiling unescaper v0.1.5
   Compiling zerofrom v0.1.5
   Compiling tracing v0.1.41
   Compiling yoke v0.7.5
   Compiling zerovec v0.10.4
   Compiling cairo-lang-syntax-codegen v2.9.2
   Compiling tinystr v0.7.6
   Compiling icu_collections v1.5.0
   Compiling icu_locid v1.5.0
   Compiling icu_provider v1.5.0
```

```
   Compiling icu_locid_transform v1.5.0
   Compiling icu_properties v1.5.1
   Compiling indexmap v2.7.1
   Compiling hashbrown v0.14.5
   Compiling toml_datetime v0.6.8
   Compiling num-bigint v0.4.6
   Compiling serde_spanned v0.6.8
   Compiling triomphe v0.1.14
   Compiling smol_str v0.2.2
   Compiling icu_normalizer v1.5.0
   Compiling toml_edit v0.22.23
   Compiling rust-analyzer-salsa v0.17.0-pre.6
   Compiling idna_adapter v1.2.0
   Compiling idna v1.0.3
   Compiling cairo-lang-utils v2.9.2
   Compiling url v2.5.4
   Compiling cairo-lang-debug v2.9.2
   Compiling toml v0.8.20
   Compiling cairo-lang-filesystem v2.9.2
   Compiling cairo-lang-diagnostics v2.9.2
   Compiling cairo-lang-syntax v2.9.2
   Compiling cairo-lang-parser v2.9.2
   Compiling snforge_scarb_plugin v0.35.0 (/scarb/registry/src/scarbs.xyz-9djtpev4jug5q/snforge_scarb_plugin
     -0.35.0)
    Finished `release` profile [optimized] target(s) in 25.54s
   Compiling lib(opus_compose) opus_compose v0.1.0 (/contracts/Scarb.toml)
   Compiling starknet-contract(opus_compose) opus_compose v0.1.0 (/contracts/Scarb.toml)
    Finished `dev` profile target(s) in 60 seconds
```

8.2 Tests Output

```
scarb test
     Running test opus_compose (snforge test)
   Compiling snforge_scarb_plugin v0.35.0
    Finished `release` profile [optimized] target(s) in 0.04s
   Compiling test(opus_compose_unittest) opus_compose v0.1.0 (/contracts/Scarb.toml)
    Finished `dev` profile target(s) in 13 seconds

Collected 24 test(s) from opus_compose package
Running 24 test(s) from src/

[PASS] opus_compose::lever::tests::test_lever::test_lever_down_insufficient_trove_yang_fail (gas: ~5937)
[PASS] opus_compose::lever::tests::test_lever::test_lever_down_unhealthy_fail (gas: ~6416)
[PASS] opus_compose::lever::tests::test_lever::test_unauthorized_lever_up_fail (gas: ~709)
[PASS] opus_compose::stabilizer::tests::test_stabilizer::test_stake_wrong_upper_tick_fail (gas: ~2792)
[PASS] opus_compose::stabilizer::tests::test_stabilizer::test_non_user_claim_fail (gas: ~3096)
[PASS] opus_compose::stabilizer::tests::test_stabilizer::test_stake_unapproved_fail (gas: ~2991)
[PASS] opus_compose::stabilizer::tests::test_stabilizer::test_stake (gas: ~3126)
[PASS] opus_compose::stabilizer::tests::test_stabilizer::test_non_user_unstake_fail (gas: ~3096)
[PASS] opus_compose::stabilizer::tests::test_stabilizer::test_claim (gas: ~5881)
[PASS] opus_compose::stabilizer::tests::test_stabilizer::test_claim_large_surplus_with_low_liquidity_no_overflow
    (gas: ~3542)
[PASS] opus_compose::stabilizer::tests::test_stabilizer::test_unstake (gas: ~3413)
[PASS] opus_compose::stabilizer::tests::test_stabilizer::test_stake_wrong_lower_tick_fail (gas: ~2791)
[PASS] opus_compose::stabilizer::tests::test_stabilizer::test_multi_users (gas: ~5511)
Tests: 24 passed, 0 failed, 0 skipped, 0 ignored, 0 filtered out
```