CAIRO SECURITY CLAN

CLOBER

SECURITY ASSESMENT REPORT

NOVEMBER 2024

Prepared for

CLOBER

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 Clober protocol.

Clober presents a new algorithm for order book DEX "LOBSTER - Limit Order Book with Segment Tree for Efficient order-matching" that enables on-chain order matching and settlement on decentralized smart contract platforms. With Clober, market participants can place limit and market orders in a fully decentralized, trustless way at a manageable cost. 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

9 points of attention, where 2 are classified as Critical, 0 is classified as High, 2 are classified as Medium,1 is classified as Low,3 are classified as Informational and 1 is 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.

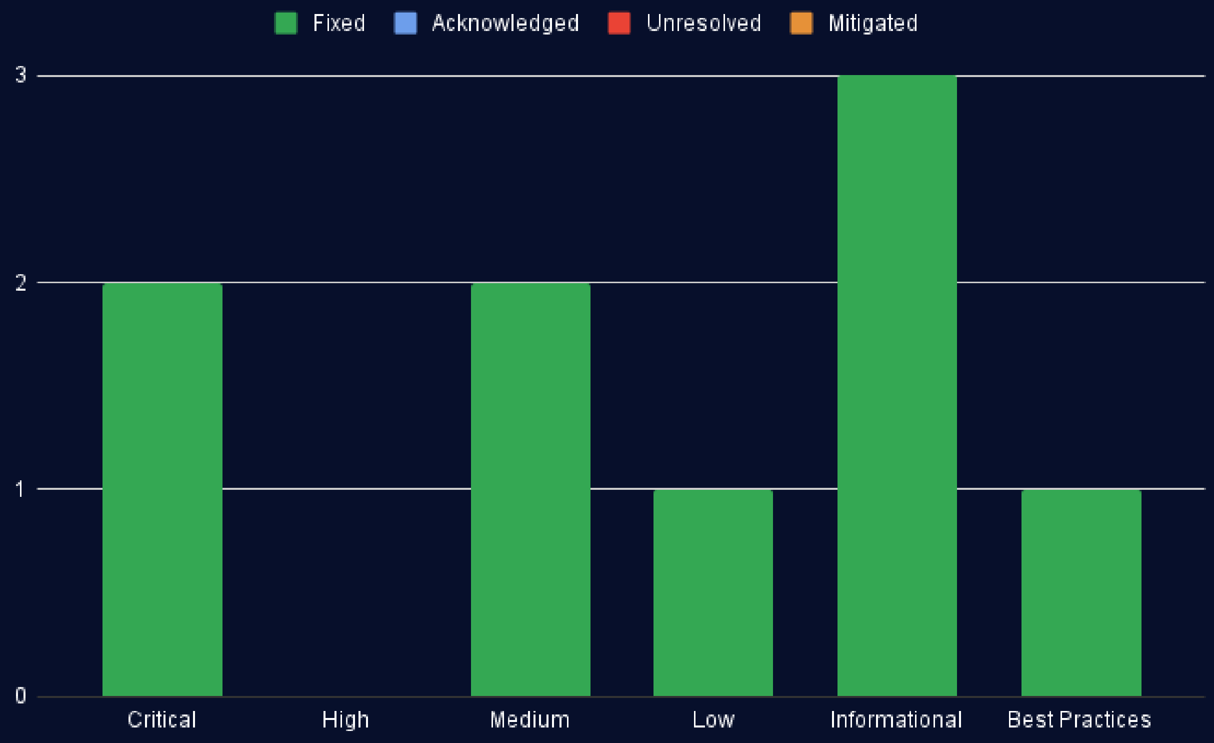

Fig 1: Distribution of issues: Critical (2), **High** (0), **Medium** (2), **Low** (1), **Informational** (3), **Best Practices** (1).
Distribution of status: Fixed (9), **Acknowledged** (0), **Mitigated** (0), **Unresolved** (0).

4 Summary of Audit

Audit Type	Security Review
Cairo Version	2.8.0
Final Report	19/11/2024
Repository	clober_cairo
Initial Commit Hash	c7ca58e30544c69a87ea39c70389b70d384394e1
Final Commit Hash	e4029f8c1f11bd19e9bbbdd30c016e6b7e1d9916
Documentation	Website documentation
Test Suite Assessment	High

4.1 Scoped Files

	Contracts
1	/src/book_manager.cairo
2	/src/controller.cairo
3	/src/libraries/book.cairo
4	/src/libraries/book_key.cairo
5	/src/libraries/fee_policy.cairo
6	/src/libraries/hooks.cairo
7	/src/libraries/hooks_list.cairo
8	/src/libraries/lockers.cairo
9	/src/libraries/order_id.cairo
10	/src/libraries/segmented_segment_tree.cairo
11	/src/libraries/storage_map.cairo
12	/src/libraries/tick.cairo
13	/src/libraries/tick_bitmap.cairo
14	/src/libraries/total_claimable_map.cairo
15	/src/utils/math.cairo
16	/src/utils/packed_felt252.cairo

4.2 Issues

	Findings	**Severity**	**Update**
1	Attacker could claim any order and steal the claimable base	Critical	Fixed
2	Attacker could cancel any order and steal the pending quote	Critical	Fixed
3	Flawed check in function `is_valid_hook_address()` permits hook without any flags set	Medium	Fixed
4	`is_base_remained` not set when `limit_price` reached in function `_spend()`	Medium	Fixed
5	Order Id packing can overflow felt252 limit	Low	Fixed
6	Breaking condition in the while loop in function `query()` might cause a revert due to overflow	Informational	Fixed
7	Unchecked return values of ERC20 transfer	Informational	Fixed
8	Confusing parameter name in function `reserves_of()`	Informational	Fixed
9	Function `lock_data()` might return incorrect `length` due of operator precedence	Best Practices	Fixed

5 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;

b) **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.

6 Issues by Severity Levels

6.1 Critical

6.1.1 Attacker could claim any order and steal the claimable base

File(s): /src/controller.cairo

Description: The Clober protocol uses a Lock Mechanism from UniSwap v4. The controller contract helps users interact with the core book manager. Users must approve the controller to manage their tokens and NFTs for actions like canceling or claiming orders.

In the book manager, the `claim()` function checks if the caller is authorized based on NFT ownership:

```
fn claim(ref self: ContractState, id: felt252, hook_data: Span<felt252>) -> u256 {
    self._check_locker();
    self
        .erc721
        ._check_authorized(
            self.erc721._owner_of(id.into()), get_caller_address(), id.into()
        );
    // ...
}
```

However, in the controller contract, anyone can call `claim()` for any order without needing approval from the NFT owner. This means an attacker could claim any order and steal the claimable base token:

```
fn claim(
    ref self: ContractState, order_id: felt252, hook_data: Span<felt252>, deadline: u64
) {
    self._check_deadline(deadline);
    let book_manager = IBookManagerDispatcher {
        contract_address: self.book_manager.read()
    };
    let mut params = ArrayTrait::new();
    Serde::serialize(@order_id, ref params);
    Serde::serialize(@hook_data, ref params);

    let mut data = ArrayTrait::new();
    Serde::serialize(@get_caller_address(), ref data);
    Serde::serialize(@Actions::Claim, ref data);
    Serde::serialize(@params, ref data);
    book_manager.lock(get_contract_address(), data.span());
}
```

Recommendation(s): Consider adding a check to ensure the caller has approval from the NFT owner before allowing them to claim an order of other users.

Status: Fixed

Update from client: Fixed in this commit.

6.1.2 Attacker could cancel any order and steal the pending quote

File(s): /src/controller.cairo

Description: The Clober protocol uses a Lock Mechanism from UniSwap v4. The controller contract helps users interact with the core book manager. Users must approve the controller to manage their tokens and NFTs for actions like canceling or claiming orders.

In the book manager, the `cancel()` function checks if the caller is authorized based on NFT ownership:

```
fn cancel(ref self: ContractState, params: CancelParams, hook_data: Span<felt252>) -> u256 {
    self._check_locker();
    self
        .erc721
        ._check_authorized(
            self.erc721._owner_of(params.id.into()), get_caller_address(), params.id.into()
        );
    // ...
}
```

However, in the controller contract, anyone can call `cancel()` for any order without needing approval from the NFT owner. This means an attacker could cancel any order and take the pending quote:

```
fn cancel(
    ref self: ContractState,
    order_id: felt252,
    left_quote_amount: u256,
    hook_data: Span<felt252>,
    deadline: u64
) {
    self._check_deadline(deadline);
    let book_manager = IBookManagerDispatcher {
        contract_address: self.book_manager.read()
    };
    let mut params = ArrayTrait::new();
    Serde::serialize(@order_id, ref params);
    Serde::serialize(@left_quote_amount, ref params);
    Serde::serialize(@hook_data, ref params);

    let mut data = ArrayTrait::new();
    Serde::serialize(@get_caller_address(), ref data);
    Serde::serialize(@Actions::Cancel, ref data);
    Serde::serialize(@params, ref data);
    book_manager.lock(get_contract_address(), data.span());
}
```

Recommendation(s): Consider adding a check to ensure the caller has approval from the NFT owner before allowing them to cancel an order.

Status: Fixed

Update from client: Fixed in this commit.

6.2 Medium

6.2.1 Flawed check in function `is_valid_hook_address()` permits hook without any flags set

File(s): /src/libraries/hooks.cairo

Description: The `is_valid_hook_address()` function in the `hooks` library is responsible for validating hook contract addresses. The current implementation checks if the address has at least one flag set by verifying that `address_felt252.into() >= Permission::AFTER_CLAIM`. However, this approach is flawed because it allows any address with bits set beyond bit 9 to pass the validation, even if none of the first 10 bits (flags) are set.

For example, an address with no bits from 0 to 9 set (flags) but with other bits set can still yield a value greater than `AFTER_CLAIM = 0x200`. This means that an address could bypass the check while having no valid permission flags set.

```
pub mod Permission {
    pub const BEFORE_OPEN: u256 = 0x1; // 1 << 0
    pub const AFTER_OPEN: u256 = 0x2; // 1 << 1
    pub const BEFORE_MAKE: u256 = 0x4; // 1 << 2
    pub const AFTER_MAKE: u256 = 0x8; // 1 << 3
    pub const BEFORE_TAKE: u256 = 0x10; // 1 << 4
    pub const AFTER_TAKE: u256 = 0x20; // 1 << 5
    pub const BEFORE_CANCEL: u256 = 0x40; // 1 << 6
    pub const AFTER_CANCEL: u256 = 0x80; // 1 << 7
    pub const BEFORE_CLAIM: u256 = 0x100; // 1 << 8
    pub const AFTER_CLAIM: u256 = 0x200; // 1 << 9
}

fn is_valid_hook_address(self: @Hooks) -> bool {
    // If a hook contract is set, it must have at least 1 flag set
    let address_felt252: felt252 = (*self).into();
    address_felt252 == 0 || address_felt252.into() >= Permission::AFTER_CLAIM // @audit Still allow hook with 0
     flag set
}
```

Recommendation(s): Consider modifying the check to

```
- address_felt252 == 0 || address_felt252.into() >= Permission::AFTER_CLAIM
+ address_felt252 == 0 || (address_felt252.into() & (0x200 - 1)) > 0
```

Status: Fixed

Update from client: Fixed in this commit.

6.2.2 `is_base_remained` not set when `limit_price` reached in function `_spend()`

File(s): /src/controller.cairo

Description: The function `_spend()` is called inside `_limit()` to spend as much quote as possible within the given price limit. If the `take_book` has insufficient liquidity or the `limit_price` is reached, the remaining quote is supposed to be used for an order in the `make_book`.

```
fn _limit(
    self: @ContractState,
    take_book_id: felt252,
    make_book_id: felt252,
    limit_price: u256,
    tick: Tick,
    mut quote_amount: u256,
    take_hook_data: Span<felt252>,
    make_hook_data: Span<felt252>
) -> (felt252, Span<felt252>) {
    let (is_quote_remained, spent_quote_amount, tokens) = self
        ._spend(take_book_id, limit_price, quote_amount, 0, take_hook_data);
    quote_amount -= spent_quote_amount;
    if is_quote_remained {
        let (order_id, _) = self._make(make_book_id, tick, quote_amount, make_hook_data);
        (order_id, tokens)
    } else {
        (0, tokens)
    }
}
```

However, there is an issue in the `_spend()` function: when the `limit_price` is reached, the loop breaks without setting `is_base_remained = true`. This prevents the function from properly handling remaining quote. As a result, if not all of the quote is spent (because the limit price was reached), the `_limit()` function does not place a new order with the remaining quote, which is the expected behavior.

```
let mut is_base_remained = false;

while max_base_amount > spent_base_amount {
    if book_manager.is_empty(book_id) {
        is_base_remained = true;
        break;
    }
    let tick = book_manager.get_highest(book_id);
    if limit_price > tick.to_price() { // @audit Not set `is_base_remained = true` before break
        break;
    }
    // ...
};
```

Recommendation(s): Consider modifying the code to set `is_base_remained = true` before breaking the loop when `limit_price` is reached.

Status: Fixed

Update from client: Fixed in this commit.

6.3 Low

6.3.1 Order Id packing can overflow `felt252` limit

File(s): /src/libraries/order_id.cairo

Description: The struct `OrderId` wanted to be packed into one `felt252` to store efficiently.

```
pub struct OrderId {
    pub book_id: felt252, // u187
    pub tick: Tick, // i24
    pub index: u64, // u40
}
```

However, in the function `encode(...)`, the variable `book_id` assertion only checks if it is lower than `2^192`, which is 192 bits long.

```
fn encode(self: OrderId) -> felt252 {
    // @audit-issue Why lower than 192 bits instead of 187?
    assert(self.book_id.into() < TWO_POW_192, 'book_id overflow');
    assert(self.index < TWO_POW_40, 'index overflow');
    let t = if self.tick.into() < 0_i32 {
        0x1000000 + self.tick.into()
    } else {
        self.tick.into()
    };
    assert(t < 0x1000000_i32, 'tick overflow');
    return self.book_id * TWO_POW_64.into() + t.into() * TWO_POW_40.into() + self.index.into();
    // @audit-issue Highest possible bit-size 192 + 40 + 24 = 256
    // @audit-issue Possible packing with max values can be higher than 252 bit felt range
}
```

This can cause possible overflow and revert during the encoding.

Recommendation(s): Consider checking `book_id` lower than expected bit size.

Status: Fixed

Update from client: Fixed in this commit.

6.4 Info

6.4.1 Breaking condition in the while loop in function `query()` might cause a revert due to overflow

File(s): /src/libraries/segmented_segment_tree.cairo

Description: In the `query()` function, a while loop is used to perform the query. The break condition is `l < 0`, where `l` starts from `L-1` and decreases by `1` in each iteration. However, since `l` is defined as an unsigned integer (u32), it cannot be negative. As a result, when `l` is decremented below 0 (i.e., `l = -1`), the loop will not break as expected. Instead, it will cause the transaction to revert due to overflow.

```
let mut l: u32 = (L - 1).into();

// @audit `l` is u32, when `l = -1`, it will be an error instead of breaking the loop
while l >= 0 {
    // ...
    l -= 1;
};
```

Recommendation(s): Use a signed integer for `l` instead of an unsigned integer.

Status: Fixed

Update from client: Fixed in this commit.

6.4.2 Unchecked return values of ERC20 transfer

File(s): /src/controller.cairo, /src/book_manager.cairo

Description: In the codebase, there are a few ERC20 transfers to move tokens between the users' wallet and book manager, controller contracts using `transfer()` and `transfer_from()`. These functions return a boolean value indicating whether the transfer succeeded or not. However, these return values are not currently checked, potentially resulting in unexpected behavior, especially if the token does not revert on failure.

```
if currency_delta.is_negative() {
    token_dispatcher
        .transfer_from( // @audit unsafe transfer
            user, book_manager.contract_address, currency_delta.abs().into()
        );
    book_manager.settle(token);
}

currency_delta = book_manager.get_currency_delta(controller_address, token);
if !currency_delta.is_negative() {
    book_manager.withdraw(token, user, currency_delta.abs().into());
}

let balance = token_dispatcher.balance_of(controller_address);
if balance > 0 {
    token_dispatcher.transfer(user, balance); // @audit unsafe transfer
}
```

Recommendation(s): Consider adding a check for the boolean return values.

Status: Fixed

Update from client: Fixed in this commit.

6.4.3 Confusing parameter name in function `reserves_of()`

File(s): /src/book_manager.cairo

Description: The function `reserves_of()` is currently designed to return the reserve of a currency in the book manager. However, it takes the `provider` address as an input parameter, which could lead to confusion as the function's purpose is related to currency reserves.

```
fn reserves_of(self: @ContractState, provider: ContractAddress) -> u256 { // @audit Should be `currency` instead
    of `provider`
    self.reserves_of.read(provider)
}
```

Recommendation(s): Consider renaming the input parameter to `currency` instead of `provider`.

Status: Fixed

Update from client: Fixed in this commit.

6.5 Best Practices

6.5.1 Function `lock_data()` might return incorrect `length` due to operator precedence

File(s): /src/libraries/lockers.cairo

Description: The current implementation of the `lock_data()` function may calculate the length incorrectly due to operator precedence. As per the Cairo documentation, the `&` operator has higher precedence than the `-` operator. This means the calculation will be evaluated as `(packed_u256 & TWO_POW_32.into()) - 1`, which is not the intended logic.

```
fn lock_data(self: @Lockers) -> (u32, u128) {
    let (address_domain, base) = self.get_base_storage_address();
    let packed: felt252 = Store::read(address_domain, base).unwrap_syscall();
    let packed_u256: u256 = packed.into();
    let length: u32 = (packed_u256 & TWO_POW_32.into() - 1).try_into().unwrap(); // @audit & will be executed
     before minus
    let non_zero_delta_count: u128 = (packed_u256 / TWO_POW_32.into()).try_into().unwrap();
    (length, non_zero_delta_count)
}
```

Recommendation(s): Consider modifying the line to add parentheses

```
- let length: u32 = (packed_u256 & TWO_POW_32.into() - 1).try_into().unwrap();
+ let length: u32 = (packed_u256 & (TWO_POW_32.into() - 1)).try_into().unwrap();
```

Status: Fixed

Update from client: Fixed in this commit.

7 Test Evaluation

7.1 Compilation Output

```
scarb build
   Compiling clober_cairo v0.1.0 (/clober/Scarb.toml)
    Finished `dev` profile target(s) in 32 seconds
```

7.2 Tests Output

```
scarb test
     Running test clober_cairo (snforge test)
   Compiling clober_cairo v0.1.0 (/clober/Scarb.toml)
    Finished `dev` profile target(s) in 37 seconds


Collected 83 test(s) from clober_cairo package
Running 83 test(s) from src/
[PASS] clober_cairo::tests::book_manager::test_admin::test_whitelist_ownership (gas: ~879)
[PASS] clober_cairo::tests::book_manager::test_admin::test_set_default_provider_ownership (gas: ~879)
[PASS] clober_cairo::tests::book_manager::test_admin::test_set_default_provider (gas: ~886)
[PASS] clober_cairo::tests::book_manager::test_admin::test_whitelist (gas: ~950)
[PASS] clober_cairo::tests::book_manager::test_open::test_open_with_invalid_fee_policy_unmatched1 (gas: ~2348)
[PASS] clober_cairo::tests::book_manager::test_make::test_make_with_invalid_tick1 (gas: ~2871)
[PASS] clober_cairo::tests::book_manager::test_claim::test_claim_nonexistent_order (gas: ~3337)
[PASS] clober_cairo::tests::book_manager::test_cancel::test_cancel_nonexistent_order (gas: ~3337)
[PASS] clober_cairo::tests::book_manager::test_cancel::test_success (gas: ~4310)
[PASS] clober_cairo::tests::book_manager::test_cancel::test_cancel_to_zero_should_burn_with_zero_claimable (gas:
     ~3500)
[PASS] clober_cairo::tests::book_manager::test_claim::test_success (gas: ~4651)
[PASS] clober_cairo::tests::book_manager::test_claim::test_claim_should_burn_with_zero_pending_order (gas: ~4084)
[PASS] clober_cairo::tests::book_manager::test_cancel::test_cancel_to_zero_with_partially_taken_order (gas:
     ~4318)
[PASS] clober_cairo::tests::book::test_take (gas: ~993)
[PASS] clober_cairo::tests::controller::test_make::test_make (gas: ~3552)
[PASS] clober_cairo::tests::book_manager::test_fee::test_maker_QP_taker_QP (gas: ~4472)
[PASS] clober_cairo::tests::book_manager::test_fee::test_maker_BP_taker_QP (gas: ~4473)
[PASS] clober_cairo::tests::book_manager::test_open::test_success (gas: ~2384)
[PASS] clober_cairo::tests::book_manager::test_fee::test_maker_QP_taker_BP (gas: ~4474)
[PASS] clober_cairo::tests::book_manager::test_make::test_make_with_invalid_book_key (gas: ~2653)
[PASS] clober_cairo::tests::book_manager::test_make::test_make_with_invalid_tick2 (gas: ~2871)
[PASS] clober_cairo::tests::book_manager::test_fee::test_maker_BP_taker_BP (gas: ~4471)
[PASS] clober_cairo::tests::book_manager::test_open::test_open_with_invalid_unit_size (gas: ~2348)
[PASS] clober_cairo::tests::book::test_make (gas: ~1409)
[PASS] clober_cairo::tests::book::test_make_with_zero_unit (gas: ~1)
[PASS] clober_cairo::tests::controller::test_limit::test_limit (gas: ~4810)
[PASS] clober_cairo::tests::book_manager::test_take::test_success (gas: ~4297)
[PASS] clober_cairo::tests::fee_policy::encode (gas: ~1)
[PASS] clober_cairo::tests::book::test_calculate_claimable_unit_not_overflow (gas: ~570)
[PASS] clober_cairo::tests::book::test_cancel_to_too_large_amount (gas: ~861)
[PASS] clober_cairo::tests::book::test_cancel (gas: ~903)
[PASS] clober_cairo::tests::controller::test_cancel::test_cancel_all (gas: ~3648)
[PASS] clober_cairo::tests::book::test_take_and_clean_tick_bitmap (gas: ~1245)
[PASS] clober_cairo::tests::packed_felt252::test_get_u62_out_of_bounds (gas: ~1)
[PASS] clober_cairo::tests::book_manager::test_take::test_take_with_invalid_book_key (gas: ~2955)
[PASS] clober_cairo::tests::packed_felt252::add_u62_overflow (gas: ~1)
[PASS] clober_cairo::tests::packed_felt252::test_update_64_out_of_bounds (gas: ~1)
[PASS] clober_cairo::tests::packed_felt252::test_update_62 (gas: ~58)
[PASS] clober_cairo::tests::book_manager::test_open::test_open_with_invalid_fee_policy_boundary1 (gas: ~2348)
[PASS] clober_cairo::tests::packed_felt252::test_add_u62_out_of_bounds (gas: ~1)
[PASS] clober_cairo::tests::packed_felt252::test_add_u62 (gas: ~65)
[PASS] clober_cairo::tests::book_manager::test_take::test_success_with_greater_max_unit (gas: ~3992)
[PASS] clober_cairo::tests::packed_felt252::test_sub_u62 (gas: ~64)
[PASS] clober_cairo::tests::packed_felt252::test_sub_u62_out_of_bounds (gas: ~1)
[PASS] clober_cairo::tests::controller::test_open::test_open (gas: ~2383)
[PASS] clober_cairo::tests::controller::test_claim::test_claim (gas: ~4393)
```

```
[PASS] clober_cairo::tests::packed_felt252::sub_u62_overflow (gas: ~1)
[PASS] clober_cairo::tests::segmented_segment_tree::test_get (gas: ~1750)
[PASS] clober_cairo::tests::tick_bitmap::test_set (gas: ~404)
[PASS] clober_cairo::tests::segmented_segment_tree::test_total (gas: ~1920)
[PASS] clober_cairo::tests::segmented_segment_tree::test_query (gas: ~1760)
[PASS] clober_cairo::tests::controller::test_take::test_take (gas: ~6218)
[PASS] clober_cairo::tests::segmented_segment_tree::test_update (gas: ~2043)
[PASS] clober_cairo::tests::total_claimable_map::test_add (gas: ~448)
[PASS] clober_cairo::tests::math::test_least_significant_bit (gas: ~5879)
[PASS] clober_cairo::tests::book_manager::test_make::test_make_with_invalid_provider (gas: ~2871)
[PASS] clober_cairo::tests::book_manager::test_fee::test_maker_QN_taker_QP (gas: ~4472)
[PASS] clober_cairo::tests::controller::test_spend::test_spend (gas: ~6284)
[PASS] clober_cairo::tests::book_manager::test_make::test_success (gas: ~3677)
[PASS] clober_cairo::tests::book::test_claim (gas: ~933)
[PASS] clober_cairo::tests::book::test_calculate_claimable_unit (gas: ~946)
[PASS] clober_cairo::tests::book_manager::test_open::test_open_with_invalid_fee_policy_unmatched2 (gas: ~2348)
[PASS] clober_cairo::tests::book_manager::test_open::test_open_with_invalid_fee_policy_unmatched4 (gas: ~2348)
[PASS] clober_cairo::tests::order_id::encode (gas: ~1)
[PASS] clober_cairo::tests::book_manager::test_open::test_open_with_invalid_fee_policy_unmatched3 (gas: ~2348)
[PASS] clober_cairo::tests::book_manager::test_open::test_open_with_invalid_fee_policy_boundary4 (gas: ~2348)
[PASS] clober_cairo::tests::fee_policy::decode (gas: ~2)
[PASS] clober_cairo::tests::book_manager::test_fee::test_maker_BN_taker_BP (gas: ~4471)
[PASS] clober_cairo::tests::book_manager::test_admin::test_delist_ownership (gas: ~950)
[PASS] clober_cairo::tests::order_id::decode (gas: ~6)
[PASS] clober_cairo::tests::book_manager::test_open::test_open_duplicated (gas: ~2571)
[PASS] clober_cairo::tests::packed_felt252::test_get_u62 (gas: ~6)
[PASS] clober_cairo::tests::fee_policy::calculate_fee (gas: ~23)
[PASS] clober_cairo::tests::book_manager::test_fee::test_maker_BP_taker_BN (gas: ~4471)
[PASS] clober_cairo::tests::book_manager::test_open::test_open_with_invalid_fee_policy_boundary2 (gas: ~2348)
[PASS] clober_cairo::tests::book_manager::test_open::test_open_with_invalid_fee_policy_negative_sum (gas: ~2348)
[PASS] clober_cairo::tests::book_manager::test_admin::test_delist (gas: ~894)
[PASS] clober_cairo::tests::book_manager::test_fee::test_maker_QP_taker_QN (gas: ~4472)
[PASS] clober_cairo::tests::book_manager::test_open::test_open_with_invalid_fee_policy_boundary3 (gas: ~2348)
[PASS] clober_cairo::tests::book_manager::test_cancel::test_cancel_auth (gas: ~4294)
[PASS] clober_cairo::tests::tick_bitmap::test_highest (gas: ~17863)
[PASS] clober_cairo::tests::tick_bitmap::test_clear (gas: ~20135)
[PASS] clober_cairo::tests::tick::test_tick_to_price (gas: ~28918)
Tests: 83 passed, 0 failed, 0 skipped, 0 ignored, 0 filtered out
```