CAIRO SECURITY
CLAN

CADDY FINANCE

SECURITY ASSESSMENT REPORT

FEBRUARY 2026

Prepared for

CADDY FINANCE

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 Scope. 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 Caddy Finance developed by Caddy Finance.

3.1 User Flows

Flow 1: Deposit Collateral

1. User calls `BitcoinVault.deposit_collateral(amount)`.
2. Vault transfers tBTC from user, deducts management fee, and sends fee to treasury.
3. Vault deposits tBTC to Vesu, borrows USDC, and forwards USDC to `trader_wallet`.
4. Vault updates per-cycle collateral and weighted-collateral accounting.

Flow 2: Request Withdrawal

1. User calls `BitcoinVault.request_withdrawal()`.
2. Vault flags the user as withdrawing for the current cycle and increments per-cycle withdrawal totals.

Flow 3: Withdraw Collateral + Yield

1. After cycle end, user calls `BitcoinVault.withdraw_collateral(cycle_id)`.
2. Vault computes yield entitlement for the cycle and transfers (collateral + yield) in tBTC.

Flow 4: Cycle End (Admin/Batched)

1. Cycle manager calls `VaultCycleManager.end_cycle()` after cycle duration elapsed.
2. Vault claims USDC yield from `YieldPool`.
3. If USDC $\geq$ debt, repay Vesu and swap profit to tBTC.
4. If USDC $<$ debt, repay partially, swap tBTC to cover debt, and potentially use treasury backstop.
5. Vault processes rollover or withdrawal transitions and starts next cycle.

Flow 5: Trader Yield Deposit

1. Trader wallet calls `YieldPool.deposit_yield(cycle_id, amount)`.
2. YieldPool records yield and transfers USDC from trader into YieldPool.
3. Vault claims yield via `YieldPool.claim_yield(cycle_id)` at cycle end.

3.2 State Machine

States

State	Description	Valid Transitions
`ActiveCycle`	`active_cycle = true`	→ `InactiveCycle` on `end_cycle()`
`InactiveCycle`	`active_cycle = false`	→ `ActiveCycle` on `start_cycle()`
`Paused`	PausableComponent set	→ `Unpaused` by owner/admin

Transition Triggers

– `ActiveCycle` → `InactiveCycle`: `VaultCycleManager.end_cycle()` sets `active_cycle = false`.

– `InactiveCycle` → `ActiveCycle`: `start_cycle()` in vault or cycle manager.

– `Paused` state: owner/vault_admin pause, or permissionless `check_health_and_pause()`.

3.3 Access Control Matrix

Function	Anyone	Owner	Vault Admin	Cycle Manager	Trader	Emergency Wallets
`deposit_collateral`	X					
`request_withdrawal`	X					
`withdraw_collateral`	X					
`yieldWithdraw`	X					
`start_cycle (Vault)`		X	X			
`end_cycle (Vault)`		X	X			
`start_cycle (Manager)`		X		X		
`end_cycle (Manager)`		X		X		
`deposit_yield (YieldPool)`					X	
`claim_yield (YieldPool)`						
`emergency_withdraw (Manager)`		X				
`request_emergency_withdrawal`						X
`pause/unpause`		X	X			

3.4 Value Flow Diagram

```
[User] --tBTC--> [BitcoinVault] --tBTC--> [Vesu Pool]
[BitcoinVault] --USDC--> [Trader Wallet]
[Trader Wallet] --USDC--> [YieldPool]
[YieldPool] --USDC--> [BitcoinVault]
[BitcoinVault] --USDC--> [Vesu Pool repay]
[BitcoinVault] --USDC--> [Avnu Router] --tBTC--> [BitcoinVault]
[BitcoinVault] --tBTC--> [User] (withdraw)
```

3.5 Critical Invariants (Preliminary)

1. **Collateral Conservation**: total user collateral tracked equals vault-held tBTC + Vesu collateral minus withdrawals.
2. **Debt Repayment**: cycle end ensures Vesu debt is repaid (or cycle marked failed with recovery rate).
3. **Access Control**: only owner/vault admin can update protocol configuration or trigger admin operations.
4. **Yield Accounting**: user yield share equals weighted collateral / total weighted collateral for cycle.
5. **Emergency Safety**: emergency withdrawal requires multi-sig approvals and wait period.
6. **Oracle Freshness**: price used for borrow and swaps is fresh and bounded in change.

3.6 Audit Focus Areas

1. Cycle end logic with negative PnL and multi-step swaps (edge cases, rounding).
2. Rollover batch processing and withdrawal-request accounting consistency.
3. Oracle price safety and slippage assumptions around Avnu swaps.
4. Access control boundaries between owner, vault_admin, and cycle managers.
5. YieldPool emergency flow and claim ordering.

3.7 Audit Overview

The audit was performed using

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

There are 7 points of attention, where 1 are classified as Critical, 6 as High, 0 as Medium, 0 as Low, 0 as Informational, 0 as Best Practices, and 0 as Undetermined. 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.

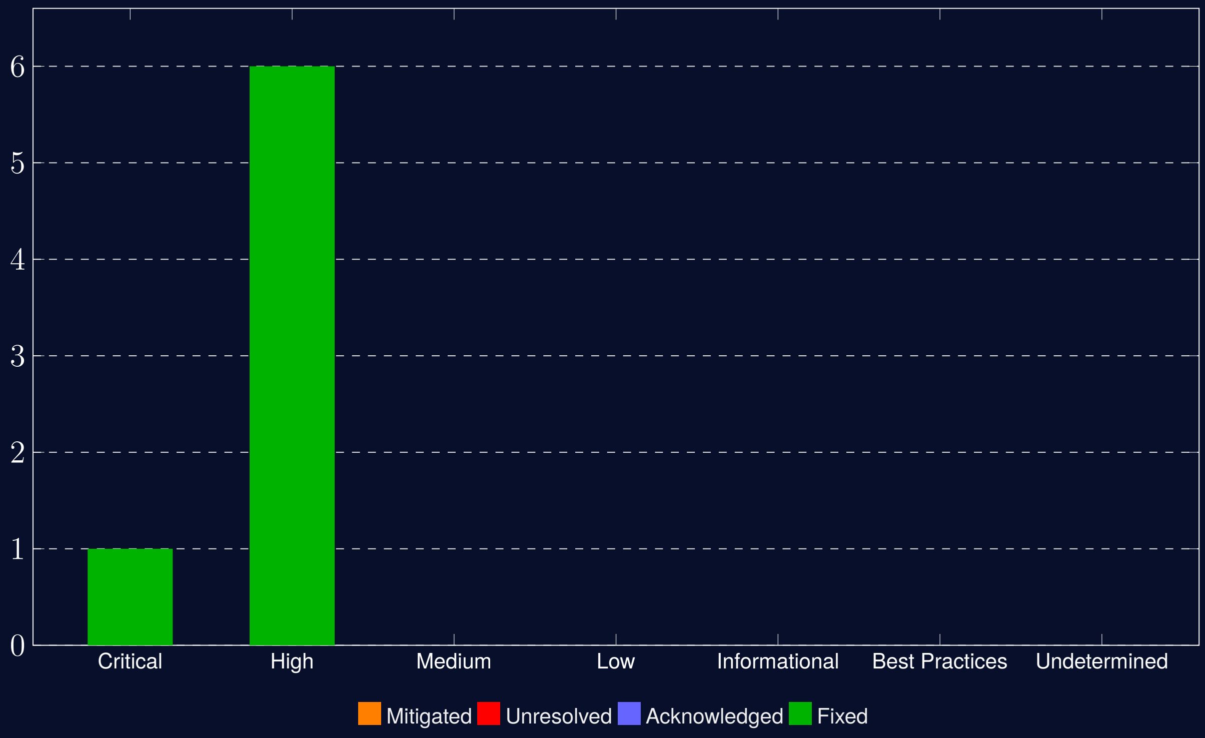

Fig 1: Distribution of issues: Critical (1), **High** (6), **Medium** (0), **Low** (0), **Informational** (0), **Best Practices** (0), **Undetermined** (0). **Distribution of status: Fixed** (7), **Acknowledged** (0), **Mitigated** (0), **Unresolved** (0).

4 System Architecture

4.1 Overview

The protocol is built around a tBTC collateral vault (`BitcoinVault`) with a separate cycle manager (`VaultCycleManager`) and a yield aggregation pool (`YieldPool`). Users deposit tBTC into the vault; the vault borrows USDC against that collateral from Vesu and forwards USDC to an off-chain trader wallet. At cycle end, USDC yield is deposited into the `YieldPool` and claimed by the vault, which repays Vesu debt and converts remaining USDC into tBTC to distribute to users.

4.2 Component Diagram

```
[User]
  | deposit/withdraw
  v
[BitcoinVault] <----> [BitcoinVaultView] (read-only)
  |  \
  |   \ claims yield
  |    v
  |  [YieldPool] <-- USDC yield from Trader
  |
  | borrow/repay/withdraw
  v
[Vesu Pool + Oracle]
  |
  | price feed
  v
[Pragma Oracle]
  |
  | swaps
  v
[Avnu Router]
```

4.3 Contract Relationships

- `BitcoinVault` holds user collateral and Vesu positions, manages cycles and user accounting.
- `VaultCycleManager` executes privileged batch processing and cycle end logic via `IVaultInternal` interface.
- `YieldPool` receives USDC yield from the trader wallet and transfers yield to the vault after cycle end.
- `BitcoinVaultView` exposes read-only getters for off-chain/UI queries.

4.4 External Dependencies

Dependency	Purpose	Trust Level
Vesu V2 Pool	Collateral/borrow/repay	High
Vesu Oracle	LTV + valuation	High
Pragma Oracle	Spot price (BTC/USD)	High
Avnu Router	USDC↔tBTC swaps	Medium
ERC20 Tokens	tBTC + USDC	High

4.5 Trust Assumptions

- Pragma oracle provides fresh BTC/USD prices within `MAX_PRICE_AGE` and bounded change thresholds.
- Vesu pool maintains collateral/borrow invariants and honors `modify_position`.
- Avnu swap returns expected token amounts (subject to slippage checks).
- Trader wallet is trusted to return yield to `YieldPool` and not maliciously delay or underfund deposits.

4.6 Upgrade Mechanisms

- `BitcoinVault` and `YieldPool` use OpenZeppelin `UpgradeableComponent` with explicit queue + timelock for upgrades.
- `VaultCycleManager` exposes `IUpgradeable`-compatible upgrade path with timelock.

5 Summary of Audit

Audit Method	Automated
Cairo Version	2.11.4
Final Report	04/02/2026
Repository	caddyfinance/Starknet-Contracts
PR #21	#21
Initial Commit Hash	89e7c189aa85f036ad9fed1d72d386fc1ad185ab
Documentation	Not Available
Test Suite Assessment	Low
Centralization Level	Highly Centralized

5.1 Scoped Files

	Contracts
1	/src/lib.cairo
2	/src/bitcoin_vault.cairo
3	/src/bitcoin_vault_view.cairo
4	/src/interfaces.cairo
5	/src/utils.cairo
6	/src/yield_pool.cairo
7	/src/vault_cycle_manager.cairo
8	/src/interfaces/IAvnu.cairo
9	/src/interfaces/IVesu.cairo
10	/src/interfaces/bitcoin_vault_core_interface.cairo
11	/src/interfaces/bitcoin_vault_interface.cairo
12	/src/interfaces/bitcoin_vault_view_interface.cairo
13	/src/interfaces/lendcomp.cairo
14	/src/interfaces/lent_debt_token_interface.cairo
15	/src/interfaces/oracle.cairo
16	/src/interfaces/vault_internal_interface.cairo
17	/src/interfaces/yield_pool_interface.cairo
18	/src/utils/ERC20Helper.cairo
19	/src/utils/constants.cairo
20	/src/utils/math.cairo
21	/src/utils/pow.cairo
22	/src/utils/reentrancy_guard.cairo
23	/src/utils/safe_decimal_math.cairo
24	/src/utils/types.cairo

5.2 Issues

	Findings	**Severity**	**Update**
1	End-cycle harvest reverts because YieldPool requires `cycle_id < current_cycle`, blocking cycle completion.	Critical	Fixed
2	Delayed withdrawals allow prior-cycle collateral to be reallocated in later rollovers.	High	Fixed
3	USDC→tBTC min-out is mis-scaled by 10,000x, allowing extreme slippage on Avnu swaps.	High	Fixed
4	Withdrawers can avoid negative-PnL losses, forcing rolling users to absorb all losses or creating a withdrawal race.	High	Fixed
5	Batched rollover recomputation under-allocates later participants.	High	Fixed
6	Cumulative price deviation is never reset for CycleManager-started cycles, causing oracle checks to eventually freeze the protocol.	High	Fixed
7	Rollover allocation ignores withdrawing users' yield, causing withdrawals to revert or underpay.	High	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 End-cycle harvest reverts because YieldPool requires `cycle_id < current_cycle`, blocking cycle completion.

File(s): /src/bitcoin_vault.cairo, /src/yield_pool.cairo, /src/vault_cycle_manager.cairo

Description: `VaultCycleManager.end_cycle()` calls `BitcoinVault.internal_harvest_yield()` before any cycle index change. `internal_harvest_yield()` passes `cycle_id = current_cycle` into `YieldPool.claim_yield()`, but `claim_yield()` requires `cycle_id < current_cycle`. Once the yield pool is configured, `end_cycle()` reverts with "Cycle not completed" and the protocol cannot finish a cycle, blocking debt repayment, rollover processing, and user withdrawals that depend on cycle completion.

Relevant code paths:

– `VaultCycleManager.end_cycle()` → `internal_harvest_yield` (vault_cycle_manager.cairo:376-410).

```
    // Harvest yield from pool
    let total_yield = vault.internal_harvest_yield();
```

– `internal_harvest_yield` forwards `current_cycle` (bitcoin_vault.cairo:977-984).

```
    fn internal_harvest_yield(ref self: ContractState) -> u256 {
    InternalFunctions::_assert_admin(ref self);
    let yield_pool = self.yield_pool.read();
    if yield_pool.is_zero() { return 0; }
    let cycle = self.current_cycle.read();
    IYieldPoolDispatcher { contract_address: yield_pool }.claim_yield(cycle)
```

– `YieldPool.claim_yield` rejects `cycle_id == current_cycle` (yield_pool.cairo:268-292).

```
    let current_cycle = IBitcoinVaultDispatcher { contract_address: vault_addr }
        .get_current_cycle();
    assert!(cycle_id < current_cycle, "Cycle␣not␣completed");
```

Recommendation(s): Align cycle-id semantics across the vault and yield pool.

Options:

– Change `internal_harvest_yield()` to accept a `cycle_id` parameter and call it with the just-completed cycle (e.g., `current_cycle - 1` after incrementing the cycle in `end_cycle()`), or

– Relax `YieldPool.claim_yield()` to allow `cycle_id == current_cycle` when the vault has marked the cycle inactive and its end time has elapsed.

Status: Fixed

Update from the client: Fixed in PR #21

7.2 High

7.2.1 Delayed withdrawals allow prior-cycle collateral to be reallocated in later rollovers.

File(s): /src/bitcoin_vault.cairo, /src/vault_cycle_manager.cairo

Description: Withdrawal requests only reserve collateral for the current cycle via `cycle_withdrawal_requested_collateral`. If a user does not withdraw promptly after cycle end, their tBTC remains in the vault balance but is no longer counted in `total_withdrawing` for subsequent cycles. At the next cycle end, `_calculate_rollover_totals` uses the full vault tBTC balance minus only the current cycle's withdrawal total. This includes prior-cycle unwithdrawn collateral, which is then allocated to rolling users in `_process_participants_range`, leaving the original withdrawer underfunded or unable to withdraw.

Relevant logic:

– `request_withdrawal()` only increments `cycle_withdrawal_requested_collateral` for the current cycle (bitcoin_vault.cairo:647-676).

```
    // Check if already requested for this cycle to avoid double-counting
    let already_requested = self.user_cycle_withdrawal_requests.read((caller, current_cycle));

    self.withdrawal_requests.write(caller, true);
    self.user_cycle_withdrawal_requests.write((caller, current_cycle), true);

    // Track withdrawal collateral incrementally to avoid unbounded loops
    // Only add to total if this is a new request for this cycle
    if !already_requested {
        let current_withdrawal_total = self.cycle_withdrawal_requested_collateral.read(current_cycle);
        self.cycle_withdrawal_requested_collateral.write(current_cycle, current_withdrawal_total +
     user_collateral);
    }
```

— `withdraw_collateral()` has no deadline; users can withdraw long after cycle end (bitcoin_vault.cairo:679-736).

```
    fn withdraw_collateral(ref self: ContractState, cycle_id: u64) {
    InternalFunctions::_non_reentrant(ref self);
    // Prevent withdrawal during end_cycle processing
    assert!(!self.end_cycle_in_progress.read(), "Cycle␣ending");
    let caller = get_caller_address();
    let current_cycle = self.current_cycle.read();
    assert!(cycle_id > 0 && cycle_id <= current_cycle, "Bad␣cycle");
    assert!(self.withdrawal_requests.read(caller), "No␣request");

    let cycle_start = self.cycle_start_times.read(cycle_id);
    let cycle_end = cycle_start + self.cycle_duration.read();
    assert!(cycle_id < current_cycle || (get_block_timestamp() >= cycle_end && !self.active_cycle.read()),
     "Cycle␣active");

    let collateral = self.user_cycle_collateral.read((caller, cycle_id));
    assert!(collateral > 0, "No␣collateral");

    // current_total >= collateral > 0 is guaranteed since user has cycle collateral
    let current_total = self.user_collateral.read(caller);

    // Calculate yield entitlement for this cycle
    let yield_entitlement = InternalFunctions::_get_user_yield_for_cycle(@self, caller, cycle_id);
    let already_withdrawn_yield = self.user_yield_withdrawn.read((caller, cycle_id));
    let yield_to_withdraw = if yield_entitlement > already_withdrawn_yield {
        yield_entitlement - already_withdrawn_yield
    } else {
        0_u256
    };

    // Total amount to transfer (collateral + yield)
    let total_to_transfer = collateral + yield_to_withdraw;

    let vault_balance = IERC20Dispatcher { contract_address: self.tbtc.read() }.balance_of(
     get_contract_address());
    assert!(vault_balance >= total_to_transfer, "Low␣bal");

    // Update collateral tracking
    self.user_collateral.write(caller, current_total - collateral);
    self.user_cycle_collateral.write((caller, cycle_id), 0);
    self.user_cycle_weighted_collateral.write((caller, cycle_id), 0);
    self.user_cycle_withdrawal_requests.write((caller, cycle_id), false);

    // Clear global withdrawal flag only if user has no remaining collateral
    let remaining_collateral = current_total - collateral;
    if remaining_collateral == 0 {
        self.withdrawal_requests.write(caller, false);
    }

    // Mark full yield entitlement as withdrawn to prevent double claims
    self.user_yield_withdrawn.write((caller, cycle_id), yield_entitlement);

    // Transfer collateral and any pending yield
    let withdrawal_success = IERC20Dispatcher { contract_address: self.tbtc.read() }.transfer(caller,
     total_to_transfer);
    assert!(withdrawal_success, "Withdraw␣fail");

    self.emit(CollateralWithdrawn { user: caller, amount: collateral, cycle_id });
    if yield_to_withdraw > 0 {
        self.emit(YieldWithdrawn { user: caller, amount: yield_to_withdraw, cycle_id });
    }
    InternalFunctions::_non_reentrant_exit(ref self);
    }
```

- _calculate_rollover_totals() reserves only total_withdrawing for the current cycle while using the full vault tBTC balance (vault_cycle_manager.cairo:1299-1323).

```
    fn _calculate_rollover_totals(self: @ContractState, current_cycle: u64) -> (u256, u256) {
    let vault = self._get_vault();
    // Get pre-computed totals - O(1) operation
    let cycle_total = vault.internal_get_cycle_total_collateral(current_cycle);
    let total_withdrawing = vault.internal_get_cycle_withdrawal_collateral(current_cycle);
    // Calculute rolling collateral (those not withdrawing)
    let total_rolling = if cycle_total > total_withdrawing {
        cycle_total - total_withdrawing
    } else {
        0
    };

    let tbtc_balance = vault.internal_get_tbtc_balance();
    // Reserve tBTC for users who are withdrawing
    let total_tbtc_for_rollovers = if tbtc_balance > total_withdrawing {
        tbtc_balance - total_withdrawing
    } else {
        0
    };
    (total_rolling, total_tbtc_for_rollovers)
    }
```

- _process_participants_range() excludes withdrawal_requests from rollovers, but does not reserve their collateral across cycles (vault_cycle_manager.cairo:1336-1356).

```
    fn _process_participants_range(
    ref self: ContractState,
    current_cycle: u64,
    new_cycle: u64,
    start: u64,
    end: u64,
    total_rolling: u256,
    total_tbtc: u256
    ) {
    let vault = self._get_vault();

    let mut i = start;
    loop {
        if i >= end {
            break;
        }

        let user = vault.internal_get_participant(current_cycle, i);
        if !vault.internal_has_withdrawal_request(user) {
            let user_collateral = vault.internal_get_user_collateral(user);
            if user_collateral > 0 && total_rolling > 0 {
                let user_share = (user_collateral * total_tbtc) / total_rolling;
                self._process_user_rollover(user, user_share, new_cycle);
            }
        }

        i += 1;
    };
    }

```

Recommendation(s): Persistently reserve pending withdrawals across cycles. Options:

- Track a global pending_withdrawal_collateral and subtract it from tbtc_balance in _calculate_rollover_totals until users withdraw.
- Enforce a withdrawal deadline (e.g., must withdraw before next cycle end), and auto-mark unclaimed collateral with a recovery rate or escrow it separately.
- Maintain per-cycle escrow balances and exclude them from all future rollovers.

Status: Fixed

Update from the client: Fixed in PR #21

7.2.2 USDC→tBTC min-out is mis-scaled by 10,000x, allowing extreme slippage on Avnu swaps.

File(s): /src/bitcoin_vault.cairo, /src/interfaces/oracle.cairo

Description: `internal_swap_usdc_to_tbtc()` computes `min_tbtc` using a 1_000_000 multiplier. With USDC at 6 decimals, tBTC at 8 decimals, and Pragma prices reported in 8 decimals, the expected multiplier is 1e10. This underestimates `min_tbtc` by 10,000x, effectively disabling slippage protection in positive-PnL swaps and allowing swaps to execute at outputs far below the oracle-implied rate.

Relevant code:

– `internal_swap_usdc_to_tbtc` uses `1_000_000_u256` (bitcoin_vault.cairo:1006-1017).

```
    fn internal_swap_usdc_to_tbtc(ref self: ContractState) {
        InternalFunctions::_assert_admin(ref self);
        let avnu = self.avnu_router.read();
        if avnu.is_zero() { return; }
        let this = get_contract_address();
        let usdc_bal = IERC20Dispatcher { contract_address: self.usdc.read() }.balance_of(this);
        if usdc_bal == 0 { return; }
        let price = InternalFunctions::get_asset_price(ref self, BTC_USD).into();
        let min_tbtc = u256_mul_div(usdc_bal * 1_000_000_u256 * 99_u256, 1_u256, 100_u256 * price, Rounding
::Floor);
        ERC20Helper::approve(self.usdc.read(), avnu, usdc_bal);
        let avnu_exchange = IAvnuExchangeDispatcher { contract_address: avnu };
        let mut routes = ArrayTrait::new();
        avnu_exchange.swap_exact_token_to(self.usdc.read(), usdc_bal, usdc_bal, self.tbtc.read(), min_tbtc,
 this, routes);
    }
```

– `_swap_usdc_to_tbtc_amount` uses `10_000_000_000_u256` (the expected scale) (bitcoin_vault.cairo:1522-1534).

```
    fn _swap_usdc_to_tbtc_amount(ref self: ContractState, usdc_amount: u256) -> u256 {
        let avnu = self.avnu_router.read();
        if avnu.is_zero() || usdc_amount == 0 { return 0; }
        let this = get_contract_address();
        let usdc_token = self.usdc.read();
        let tbtc_token = self.tbtc.read();
        let tbtc_before = IERC20Dispatcher { contract_address: tbtc_token }.balance_of(this);
        let price = Self::get_asset_price(ref self, BTC_USD).into();
        let min_tbtc = u256_mul_div(usdc_amount * 10_000_000_000_u256 * 99_u256, 1_u256, 100_u256 * price,
 Rounding::Floor);
        ERC20Helper::approve(usdc_token, avnu, usdc_amount);
        let avnu_exchange = IAvnuExchangeDispatcher { contract_address: avnu };
        let routes = ArrayTrait::new();
        let swap_success = avnu_exchange.swap_exact_token_to(usdc_token, usdc_amount, usdc_amount,
 tbtc_token, min_tbtc, this, routes);
        assert!(swap_success, "Swap␣fail");
        let tbtc_after = IERC20Dispatcher { contract_address: tbtc_token }.balance_of(this);
        if tbtc_after > tbtc_before { tbtc_after - tbtc_before } else { 0 }
    }
```

– Pragma oracle interface documents 8-decimal USD pricing (oracle.cairo:20-24).

```
    #[starknet::interface]
    pub trait IPriceOracle<TContractState> {
    /// Get the price of the token in USD with 8 decimals.
    fn get_price(self: @TContractState, token: ContractAddress) -> felt252;
    /// Get the price of the token in USD with 8 decimals and update timestamp.
    fn get_price_with_time(self: @TContractState, token: ContractAddress) -> PriceWithUpdateTime;
    }
```

Recommendation(s): Align `internal_swap_usdc_to_tbtc` with the correct decimal scaling. Replace the multiplier with `10_000_000_000_u256` (or derive it from token/price decimals) and keep the 1% slippage factor. Example fix:

```
let min_tbtc = u256_mul_div(usdc_bal * 10_000_000_000_u256 * 99_u256, 1_u256, 100_u256 * price, Rounding::Floor);
```

Also consider using a shared helper to avoid divergent scaling between `_swap_usdc_to_tbtc_amount` and `internal_swap_usdc_to_tbtc`.

Status: Fixed

Update from the client: Fixed in PR #21

7.2.3 Withdrawers can avoid negative-PnL losses, forcing rolling users to absorb all losses or creating a withdrawal race.

File(s): /src/bitcoin_vault.cairo, /src/vault_cycle_manager.cairo

Description: When a cycle ends with negative PnL, `_handle_negative_pnl` repays debt using collateral, then withdraws remaining tBTC and sets `cycle_yield` to 0 without marking the cycle failed or reducing withdrawing users' collateral. Losses are only applied to users who roll, via `_process_cycle_transitions` and the rollover share calculation, while users who requested withdrawal are excluded from rollover and can withdraw their full `user_cycle_collateral` as long as the vault has balance.

Relevant logic:

- `_handle_negative_pnl` repays debt, withdraws collateral, and sets `cycle_yield` to 0 without setting a recovery rate (vault_cycle_manager.cairo:1252).

```
     vault.internal_set_cycle_yield(current_cycle, 0);
```

- `_calculate_rollover_totals` reserves full `total_withdrawing` and computes `total_tbtc_for_rollovers` from the remaining vault balance (vault_cycle_manager.cairo:1299-1323).

```
    fn _calculate_rollover_totals(self: @ContractState, current_cycle: u64) -> (u256, u256) {
    let vault = self._get_vault();

    // Get pre-computed totals - O(1) operation
    let cycle_total = vault.internal_get_cycle_total_collateral(current_cycle);
    let total_withdrawing = vault.internal_get_cycle_withdrawal_collateral(current_cycle);

    // Calculate rolling collateral (those not withdrawing)
    let total_rolling = if cycle_total > total_withdrawing {
        cycle_total - total_withdrawing
    } else {
        0
    };

    let tbtc_balance = vault.internal_get_tbtc_balance();
    // Reserve tBTC for users who are withdrawing
    let total_tbtc_for_rollovers = if tbtc_balance > total_withdrawing {
        tbtc_balance - total_withdrawing
    } else {
        0
    };

    (total_rolling, total_tbtc_for_rollovers)
    }
```

- `withdraw_collateral` pays out full `user_cycle_collateral` if vault balance is sufficient (bitcoin_vault.cairo:679-736).

```
    let vault_balance = IERC20Dispatcher { contract_address: self.tbtc.read() }.balance_of(
     get_contract_address());
    assert!(vault_balance >= total_to_transfer, "Low␣bal");
```

Recommendation(s): Align `internal_swap_usdc_to_tbtc` with the correct decimal scaling. Replace the multiplier with `10_000_000_000_u256` (or derive it from token/price decimals) and keep the 1% slippage factor. Example fix:

```
let min_tbtc = u256_mul_div(usdc_bal * 10_000_000_000_u256 * 99_u256, 1_u256, 100_u256 * price, Rounding::Floor);
```

Also consider using a shared helper to avoid divergent scaling between `_swap_usdc_to_tbtc_amount` and `internal_swap_usdc_to_tbtc`.

Status: Fixed

Update from the client: Fixed in PR #21

7.2.4 Batched rollover recomputation under-allocates later participants.

File(s): /src/vault_cycle_manager.cairo

Description: When participant count exceeds `MAX_PARTICIPANTS_PER_BATCH`, rollover processing is batched. `process_next_cycle_transition_batch` recomputes (total_rolling, total_tbtc_for_rollovers) each batch, using the current vault `tbtc_balance`.

Earlier batches deposit tBTC into Vesu (reducing the vault balance), but `total_rolling` remains based on the original cycle totals. As a result, later batches compute a smaller `total_tbtc_for_rollovers`, and the per-user share in `_process_participants_range` is reduced, under-allocating collateral to later participants.

Recommendation(s): Compute rollover totals once at batch initialization and persist them in storage for reuse across all batch calls. Do not recompute using the mutated `tbtc_balance` after prior rollovers.

Status: Fixed

Update from the client: Fixed in PR #21

7.2.5 Cumulative price deviation is never reset for CycleManager-started cycles, causing oracle checks to eventually freeze the protocol.

File(s): /src/bitcoin_vault.cairo

Description: `cycle_reference_price` is only set in the constructor and in `BitcoinVault.start_cycle`.
The production flow uses `VaultCycleManager.start_cycle`, which calls `internal_start_new_cycle` — this does not update `cycle_reference_price`. As a result, the cumulative deviation check in `get_asset_price` stays anchored to the initial price forever. Once BTC moves more than `max_cumulative_deviation_bps` (50%) from the initial price, all price-dependent actions revert ("Cumulative"), freezing deposits/borrows/swaps even across new cycles.

Recommendation(s): Reset `cycle_reference_price` when starting a new cycle via the internal path.
A minimal fix is to update `internal_start_new_cycle` to set `cycle_reference_price` to `last_price` (or freshly queried oracle price) before returning.

Status: Fixed

Update from the client: Fixed in PR #21

7.2.6 Rollover allocation ignores withdrawing users' yield, causing withdrawals to revert or underpay.

File(s): /src/bitcoin_vault.cairo, /src/vault_cycle_manager.cairo

Description: `_calculate_rollover_totals` reserves only the withdrawing collateral when calculating `total_tbtc_for_rollovers`. However, at end-cycle the vault's tBTC balance includes collateral + yield. This causes rollovers to consume yield that belongs to withdrawing users. When the withdrawing user later calls `withdraw_collateral`, the vault balance is insufficient and the call reverts ("Low bal"), or the user receives less than their entitled collateral + yield.

Recommendation(s): Exclude withdrawing users' yield from rollover allocation. One approach is to reserve `withdrawing_collateral + withdrawing_yield` in `_calculate_rollover_totals`, or to compute rollovers using only collateral and keep yield distribution separate.

This may require tracking total withdrawing yield during end-cycle (or explicitly withholding all yield until withdrawal claims are processed).

Status: Fixed

Update from the client: Fixed in PR #21

8 Compilation Evaluation

8.1 Compilation Output

```
warn: ‘edition‘ field not set in ‘[package]‘ section for package ‘pragma_lib‘
   Compiling lib(caddy_finance_contracts) caddy_finance_contracts v0.1.0 (/home/erim/caddy-finance-final-audit/
     scope/caddy_finance/Scarb.toml)
warn: Unused import: ‘caddy_finance_contracts::mocks::mock_vesu_pool::MockVesuPool::get_contract_address‘
 --> /home/erim/caddy-finance-final-audit/scope/caddy_finance/src/mocks/mock_vesu_pool.cairo:49:37
    use starknet::{ContractAddress, get_contract_address};
                                    ^^^^^^^^^^^^^^^^^^^^


   Compiling starknet-contract(caddy_finance_contracts) caddy_finance_contracts v0.1.0 (/home/erim/caddy-finance-
     final-audit/scope/caddy_finance/Scarb.toml)
warn: Unused import: ‘caddy_finance_contracts::mocks::mock_vesu_pool::MockVesuPool::get_contract_address‘
 --> /home/erim/caddy-finance-final-audit/scope/caddy_finance/src/mocks/mock_vesu_pool.cairo:49:37
    use starknet::{ContractAddress, get_contract_address};
                                    ^^^^^^^^^^^^^^^^^^^^


    Finished ‘dev‘ profile target(s) in 12 seconds
```

8.2 Tests Output

```
warn: ‘edition‘ field not set in ‘[package]‘ section for package ‘pragma_lib‘
warn: ‘edition‘ field not set in ‘[package]‘ section for package ‘pragma_lib‘
   Compiling test(caddy_finance_contracts_unittest) caddy_finance_contracts v0.1.0 (/home/erim/caddy-finance-final-audit/scope/caddy_finance/Scarb.toml)
warn: Unused import: ‘caddy_finance_contracts::mocks::mock_vesu_pool::MockVesuPool::get_contract_address‘
 --> /home/erim/caddy-finance-final-audit/scope/caddy_finance/src/mocks/mock_vesu_pool.cairo:49:37
    use starknet::{ContractAddress, get_contract_address};
                                    ^^^^^^^^^^^^^^^^^^^^


   Compiling test(caddy_finance_contracts_integrationtest) caddy_finance_contracts_integrationtest v0.1.0 (/home/erim/caddy-finance-final-audit/scope/caddy_finance/Scarb.toml)
warn: Unused import: ‘caddy_finance_contracts_integrationtest::test_contract::IMockVesuPoolDispatcher‘
 --> /home/erim/caddy-finance-final-audit/scope/caddy_finance/tests/test_contract.cairo:12:54
use caddy_finance_contracts::mocks::mock_vesu_pool::{IMockVesuPoolDispatcher, IMockVesuPoolDispatcherTrait};
                                                     ^^^^^^^^^^^^^^^^^^^^^^^

warn: Unused import: ‘caddy_finance_contracts_integrationtest::test_contract::IMockVesuPoolDispatcherTrait‘
 --> /home/erim/caddy-finance-final-audit/scope/caddy_finance/tests/test_contract.cairo:12:79
use caddy_finance_contracts::mocks::mock_vesu_pool::{IMockVesuPoolDispatcher, IMockVesuPoolDispatcherTrait};
                                                                              ^^^^^^^^^^^^^^^^^^^^^^^^^^^^

warn: Usage of deprecated feature ‘"deprecated-starknet-consts"‘ with no ‘#[feature("deprecated-starknet-consts")]‘ attribute. Note: "Use␣‘TryInto::try_into‘␣in␣const␣context␣instead."
 --> /home/erim/caddy-finance-final-audit/scope/caddy_finance/tests/test_contract.cairo:22:33
use starknet::{ContractAddress, contract_address_const, get_block_timestamp};
                                ^^^^^^^^^^^^^^^^^^^^^^


    Finished ‘dev‘ profile target(s) in 9 seconds


Collected 142 test(s) from caddy_finance_contracts package
Running 134 test(s) from tests/
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_deployment_and_getters (l1_gas: ~0, l1_data_gas: ~864, l2_gas: ~1070720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_vault_paused_state_after_unpause (l1_gas: ~0, l1_data_gas: ~1824, l2_gas: ~1101440)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_paused_getter (l1_gas: ~0, l1_data_gas: ~1824, l2_gas: ~1421440)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_has_cycle_withdrawal_request_default_false (l1_gas: ~0, l1_data_gas: ~1824, l2_gas: ~670720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_bitcoin_vault_set_trader_wallet (l1_gas: ~0, l1_data_gas: ~1824, l2_gas: ~931200)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_get_vesu_pool_address (l1_gas: ~0, l1_data_gas: ~1824, l2_gas: ~670720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_get_fee_basis_points (l1_gas: ~0, l1_data_gas: ~1824, l2_gas: ~790720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_vault_paused_state_default (l1_gas: ~0, l1_data_gas: ~1824, l2_gas: ~670720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_get_management_fee_bps_default (l1_gas: ~0, l1_data_gas: ~1824, l2_gas: ~670720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_minimum_deposit_and_borrow (l1_gas: ~0, l1_data_gas: ~7488, l2_gas: ~9380800)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_multiple_deposits_same_cycle_yield_distribution_mainnet (l1_gas: ~0, l1_data_gas: ~7584, l2_gas: ~16576000)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_has_withdrawal_request_default_false (l1_gas: ~0, l1_data_gas: ~1824, l2_gas: ~670720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_bitcoin_vault_pause_by_non_owner (l1_gas: ~0, l1_data_gas: ~1824, l2_gas: ~710720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_get_vesu_oracle (l1_gas: ~0, l1_data_gas: ~1824, l2_gas: ~670720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_request_withdrawal_no_collateral (l1_gas: ~0, l1_data_gas: ~1824, l2_gas: ~710720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_deposit_exactly_min_deposit_amount (l1_gas: ~0, l1_data_gas: ~7488, l2_gas: ~8940800)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_request_withdrawal_functional_no_collateral (l1_gas: ~0, l1_data_gas: ~1824, l2_gas: ~710720)
```

```
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_has_cycle_withdrawal_request (l1_gas: ~0,
    l1_data_gas: ~7968, l2_gas: ~9561280)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_get_cycle_collateral_recovery_rate_default (
    l1_gas: ~0, l1_data_gas: ~1824, l2_gas: ~670720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_get_cycle_duration (l1_gas: ~0, l1_data_gas:
    ~1824, l2_gas: ~630720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_get_cycle_failed_default_false (l1_gas: ~0,
    l1_data_gas: ~1824, l2_gas: ~670720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_set_yield_pool_zero_address (l1_gas: ~0,
    l1_data_gas: ~1824, l2_gas: ~710720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_start_cycle (l1_gas: ~0, l1_data_gas: ~2976,
    l2_gas: ~2062400)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_set_treasury_zero_address (l1_gas: ~0,
    l1_data_gas: ~1824, l2_gas: ~710720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_emergency_wallet_getters (l1_gas:
    ~0, l1_data_gas: ~768, l2_gas: ~710720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_vault_paused_state_after_pause (l1_gas: ~0,
    l1_data_gas: ~1920, l2_gas: ~926080)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_get_cycle_yield (l1_gas: ~0,
    l1_data_gas: ~768, l2_gas: ~510720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_start_cycle_unauthorized (l1_gas: ~0,
    l1_data_gas: ~2784, l2_gas: ~1481920)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_set_treasury_fee_bps_to_zero (l1_gas: ~0,
    l1_data_gas: ~1728, l2_gas: ~950720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_user_multiple_deposits_same_cycle (l1_gas:
    ~0, l1_data_gas: ~7488, l2_gas: ~13486080)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_deposit_below_min_deposit_amount (l1_gas: ~0,
     l1_data_gas: ~6048, l2_gas: ~4555520)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_get_emergency_wallets (l1_gas: ~0,
     l1_data_gas: ~768, l2_gas: ~710720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_pause_unpause (l1_gas: ~0,
    l1_data_gas: ~768, l2_gas: ~981440)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_withdraw_from_non_failed_cycle (l1_gas: ~0,
    l1_data_gas: ~1920, l2_gas: ~710720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_get_cycle_recovery_rate_default_zero (l1_gas:
     ~0, l1_data_gas: ~1824, l2_gas: ~670720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_request_emergency_withdrawal (
    l1_gas: ~0, l1_data_gas: ~960, l2_gas: ~976320)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_cancel_emergency_withdrawal (
    l1_gas: ~0, l1_data_gas: ~768, l2_gas: ~1812160)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_pause_unauthorized (l1_gas: ~0,
    l1_data_gas: ~768, l2_gas: ~550720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_multiple_emergency_approvals (
    l1_gas: ~0, l1_data_gas: ~1056, l2_gas: ~1201920)
[PASS] caddy_finance_contracts_integrationtest::test_contract::
    test_yield_pool_request_emergency_withdrawal_unauthorized (l1_gas: ~0, l1_data_gas: ~768, l2_gas: ~550720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::
    test_multiple_users_different_deposit_times_yield_distribution_mainnet (l1_gas: ~0, l1_data_gas: ~8064,
    l2_gas: ~16896000)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_set_trader_wallet (l1_gas: ~0,
    l1_data_gas: ~768, l2_gas: ~750720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_get_cycle_total_debt_no_deposits (l1_gas: ~0,
     l1_data_gas: ~1824, l2_gas: ~670720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_set_emergency_wait_period (l1_gas:
     ~0, l1_data_gas: ~768, l2_gas: ~750720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_set_vault (l1_gas: ~0, l1_data_gas
    : ~864, l2_gas: ~750720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_set_trader_wallet_unauthorized (
    l1_gas: ~0, l1_data_gas: ~768, l2_gas: ~550720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_set_usdc_unauthorized (l1_gas: ~0,
     l1_data_gas: ~768, l2_gas: ~550720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_deposit_with_zero_management_fee (l1_gas: ~0,
     l1_data_gas: ~7296, l2_gas: ~8749120)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_end_cycle_when_not_active (l1_gas: ~0,
    l1_data_gas: ~1824, l2_gas: ~710720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_get_user_cycle_collateral_no_deposit (l1_gas:
     ~0, l1_data_gas: ~1824, l2_gas: ~670720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_get_next_cycle_duration (l1_gas: ~0,
    l1_data_gas: ~1824, l2_gas: ~630720)
```

```
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_set_vault_not_owner (l1_gas: ~0,
    l1_data_gas: ~2592, l2_gas: ~1061440)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_unpause_unauthorized (l1_gas: ~0,
    l1_data_gas: ~864, l2_gas: ~846080)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_get_user_cycle_weighted_collateral (l1_gas:
    ~0, l1_data_gas: ~1824, l2_gas: ~670720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_get_user_cycle_collateral_no_deposits (l1_gas
    : ~0, l1_data_gas: ~1824, l2_gas: ~670720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_end_cycle_too_early (l1_gas: ~0, l1_data_gas:
     ~3072, l2_gas: ~2022400)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_end_cycle_unauthorized (l1_gas: ~0,
    l1_data_gas: ~3072, l2_gas: ~2302400)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_withdraw_future_cycle (l1_gas: ~0,
    l1_data_gas: ~1920, l2_gas: ~710720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_withdraw_no_yield (l1_gas: ~0,
    l1_data_gas: ~1920, l2_gas: ~710720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_get_current_cycle_before_any_cycle (l1_gas:
    ~0, l1_data_gas: ~1824, l2_gas: ~630720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_bitcoin_vault_set_treasury (l1_gas: ~0,
    l1_data_gas: ~1824, l2_gas: ~931200)
[PASS] caddy_finance_contracts_integrationtest::test_contract::
    test_internal_get_participants_count_no_participants (l1_gas: ~0, l1_data_gas: ~1824, l2_gas: ~670720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::
    test_withdraw_collateral_functional_when_cycle_active (l1_gas: ~0, l1_data_gas: ~3168, l2_gas: ~2342400)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_set_vault_success (l1_gas: ~0,
    l1_data_gas: ~2688, l2_gas: ~1261440)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_start_cycle_when_active (l1_gas: ~0,
    l1_data_gas: ~2976, l2_gas: ~1782400)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_three_users_deposit_same_cycle (l1_gas: ~0,
    l1_data_gas: ~8448, l2_gas: ~19534720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_set_max_price_change_bps (l1_gas: ~0,
    l1_data_gas: ~1824, l2_gas: ~830720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::
    test_get_cycle_total_weighted_collateral_no_deposits (l1_gas: ~0, l1_data_gas: ~1824, l2_gas: ~670720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_internal_set_management_fee_by_owner (l1_gas:
     ~0, l1_data_gas: ~1824, l2_gas: ~950720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_bitcoin_vault_set_yield_pool (l1_gas: ~0,
    l1_data_gas: ~2688, l2_gas: ~1321920)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_bitcoin_vault_unpause_by_non_owner (l1_gas:
    ~0, l1_data_gas: ~1920, l2_gas: ~966080)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_emergency_approval_count (l1_gas:
    ~0, l1_data_gas: ~768, l2_gas: ~510720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_get_user_total_collateral_no_deposits (l1_gas
    : ~0, l1_data_gas: ~1824, l2_gas: ~670720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_set_trader_wallet_zero_address (l1_gas: ~0,
    l1_data_gas: ~1824, l2_gas: ~710720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_get_vesu_addresses (l1_gas: ~0, l1_data_gas:
    ~1824, l2_gas: ~750720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_default_cycle_duration (l1_gas: ~0,
    l1_data_gas: ~1824, l2_gas: ~630720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_get_user_collateral_for_cycle_no_deposit (
    l1_gas: ~0, l1_data_gas: ~1824, l2_gas: ~670720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_get_cycle_yield_no_cycle (l1_gas: ~0,
    l1_data_gas: ~1824, l2_gas: ~670720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_get_cycle_total_collateral_no_deposits (
    l1_gas: ~0, l1_data_gas: ~1824, l2_gas: ~670720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_withdraw_success (l1_gas: ~0,
    l1_data_gas: ~7776, l2_gas: ~11283200)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_constructor_initialization (l1_gas: ~0,
    l1_data_gas: ~1824, l2_gas: ~1350720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::
    test_withdraw_collateral_mainnet_fork_no_request_made (l1_gas: ~0, l1_data_gas: ~7680, l2_gas: ~11759040)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_start_cycle_without_avnu_router (l1_gas: ~0,
    l1_data_gas: ~2688, l2_gas: ~1241920)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_explorer_scenario_mainnet (l1_gas: ~0,
    l1_data_gas: ~7584, l2_gas: ~11319040)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_withdraw_collateral_after_cycle_end (l1_gas:
    ~0, l1_data_gas: ~7680, l2_gas: ~13251200)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_has_withdrawal_request_getter (l1_gas: ~0,
    l1_data_gas: ~1824, l2_gas: ~670720)
```

```
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_bitcoin_vault_set_treasury_unauthorized (
    l1_gas: ~0, l1_data_gas: ~1824, l2_gas: ~710720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_is_active_cycle_before_start (l1_gas: ~0,
    l1_data_gas: ~1824, l2_gas: ~670720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_set_management_fee_bps_to_zero (l1_gas: ~0,
    l1_data_gas: ~1728, l2_gas: ~950720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_emergency_wait_period (l1_gas: ~0,
     l1_data_gas: ~768, l2_gas: ~510720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_set_usdc (l1_gas: ~0, l1_data_gas:
     ~768, l2_gas: ~811200)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_bitcoin_vault_set_trader_wallet_unauthorized
    (l1_gas: ~0, l1_data_gas: ~1824, l2_gas: ~710720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_set_cycle_duration_to_min (l1_gas: ~0,
    l1_data_gas: ~1824, l2_gas: ~910720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_get_user_cycle_weighted_collateral_no_deposit
     (l1_gas: ~0, l1_data_gas: ~1824, l2_gas: ~670720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_get_cycle_total_weighted_collateral (l1_gas:
    ~0, l1_data_gas: ~1824, l2_gas: ~670720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_internal_set_treasury_fee_by_owner (l1_gas:
    ~0, l1_data_gas: ~1824, l2_gas: ~950720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_set_vault_unauthorized (l1_gas:
    ~0, l1_data_gas: ~768, l2_gas: ~550720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_get_treasury_fee_bps_default (l1_gas: ~0,
    l1_data_gas: ~1824, l2_gas: ~670720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_set_treasury_fee_bps_to_max (l1_gas: ~0,
    l1_data_gas: ~1824, l2_gas: ~950720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_start_cycle_without_yield_pool (l1_gas: ~0,
    l1_data_gas: ~1920, l2_gas: ~830720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_get_cycle_start_time_before_start (l1_gas:
    ~0, l1_data_gas: ~1824, l2_gas: ~670720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_withdraw_collateral_mainnet_fork (l1_gas: ~0,
     l1_data_gas: ~8064, l2_gas: ~11739520)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_get_cycle_failed_due_to_insufficient_funds (
    l1_gas: ~0, l1_data_gas: ~1824, l2_gas: ~670720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_set_cycle_duration_to_max (l1_gas: ~0,
    l1_data_gas: ~1824, l2_gas: ~910720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_claim_yield_vault_not_set (l1_gas:
     ~0, l1_data_gas: ~864, l2_gas: ~550720)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_deposit_yield_vault_not_set (
    l1_gas: ~0, l1_data_gas: ~1120, l2_gas: ~1431680)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_get_vesu_collateral (l1_gas: ~0, l1_data_gas:
     ~3200, l2_gas: ~2832640)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_get_health_factor (l1_gas: ~0, l1_data_gas:
    ~3200, l2_gas: ~2992640)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_deposit_yield_unauthorized_caller
    (l1_gas: ~0, l1_data_gas: ~4160, l2_gas: ~3263360)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_claim_yield_no_yield (l1_gas: ~0,
    l1_data_gas: ~5760, l2_gas: ~3843840)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_claim_yield_transfer_succeeds (
    l1_gas: ~0, l1_data_gas: ~5760, l2_gas: ~7773760)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_deposit_yield_zero_amount_succeeds
     (l1_gas: ~0, l1_data_gas: ~5568, l2_gas: ~5550400)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_deposit_yield_successful (l1_gas:
    ~0, l1_data_gas: ~5760, l2_gas: ~6622080)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_start_cycle_by_vault_admin (l1_gas: ~0,
    l1_data_gas: ~4448, l2_gas: ~3904320)
[PASS] caddy_finance_contracts_integrationtest::test_contract::
    test_yield_pool_claim_yield_no_yield_available_panics (l1_gas: ~0, l1_data_gas: ~5856, l2_gas: ~7613760)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_claim_yield_successful_happy_path
    (l1_gas: ~0, l1_data_gas: ~5760, l2_gas: ~8013760)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_is_active_cycle_after_start (l1_gas: ~0,
    l1_data_gas: ~4448, l2_gas: ~3784320)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_deposit_collateral_zero_amount (l1_gas: ~0,
    l1_data_gas: ~4640, l2_gas: ~4064320)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_claim_yield_when_paused (l1_gas:
    ~0, l1_data_gas: ~6048, l2_gas: ~6917440)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_claim_yield_unauthorized_caller (
    l1_gas: ~0, l1_data_gas: ~5952, l2_gas: ~6542080)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_deposit_while_paused (l1_gas: ~0, l1_data_gas
    : ~4832, l2_gas: ~4840640)
```

```
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_deposit_yield_transfer_succeeds (l1_gas: ~0, l1_data_gas: ~5760, l2_gas: ~6302080)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_yield_pool_deposit_yield_when_paused (l1_gas: ~0, l1_data_gas: ~4352, l2_gas: ~4239680)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_bitcoin_vault_deposit_collateral_when_paused (l1_gas: ~0, l1_data_gas: ~4928, l2_gas: ~5200640)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_bitcoin_vault_deposit_collateral_no_active_cycle (l1_gas: ~0, l1_data_gas: ~4448, l2_gas: ~4364800)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_deposit_just_above_min_deposit_amount (l1_gas: ~0, l1_data_gas: ~7488, l2_gas: ~8860800)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_deposit_collateral_succeeds_with_valid_amount (l1_gas: ~0, l1_data_gas: ~7488, l2_gas: ~8860800)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_deposit_collateral_mainnet_fork (l1_gas: ~0, l1_data_gas: ~7488, l2_gas: ~10020800)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_end_cycle_by_vault_admin (l1_gas: ~0, l1_data_gas: ~7392, l2_gas: ~9420800)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_consecutive_cycles (l1_gas: ~0, l1_data_gas: ~7584, l2_gas: ~10180800)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_failed_cycle_collateral_distribution (l1_gas: ~0, l1_data_gas: ~7680, l2_gas: ~17237440)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_get_cycle_start_time_after_start (l1_gas: ~0, l1_data_gas: ~4448, l2_gas: ~3904320)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_deposit_collateral_reverts_when_borrow_fails (l1_gas: ~0, l1_data_gas: ~4832, l2_gas: ~4905280)
[PASS] caddy_finance_contracts_integrationtest::test_contract::test_deposit_after_cycle_end_time (l1_gas: ~0, l1_data_gas: ~4832, l2_gas: ~32985280)
Running 8 test(s) from src/
[PASS] caddy_finance_contracts::utils::pow::tests::test_ten_pow (l1_gas: ~0, l1_data_gas: ~0, l2_gas: ~40000)
[PASS] caddy_finance_contracts::utils::pow::tests::test_ten_pow_overflow (l1_gas: ~0, l1_data_gas: ~0, l2_gas: ~40000)
[PASS] caddy_finance_contracts::utils::safe_decimal_math::tests::test_mul_decimals (l1_gas: ~0, l1_data_gas: ~0, l2_gas: ~80000)
[PASS] caddy_finance_contracts::utils::safe_decimal_math::tests::test_mul_overflow (l1_gas: ~0, l1_data_gas: ~0, l2_gas: ~80000)
[PASS] caddy_finance_contracts::utils::safe_decimal_math::tests::test_div (l1_gas: ~0, l1_data_gas: ~0, l2_gas: ~80000)
[PASS] caddy_finance_contracts::utils::safe_decimal_math::tests::test_mul (l1_gas: ~0, l1_data_gas: ~0, l2_gas: ~80000)
[PASS] caddy_finance_contracts::utils::safe_decimal_math::tests::test_mul_decimals_overflow (l1_gas: ~0, l1_data_gas: ~0, l2_gas: ~80000)
[PASS] caddy_finance_contracts::utils::safe_decimal_math::tests::test_div_decimals (l1_gas: ~0, l1_data_gas: ~0, l2_gas: ~80000)
Tests: 142 passed, 0 failed, 0 skipped, 0 ignored, 0 filtered out

Latest block number = 6363735 for url = http://51.195.57.196:6060/v0_8
```