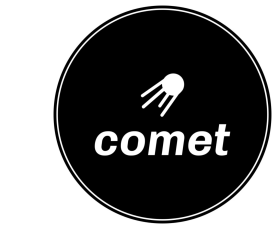


Comet

Audit

Presented by:

OtterSec contact@osec.io

Nicola Vella nick0ve@osec.io

Andreas Mantzoutas andreas@osec.io

Contents

01 | Executive Summary

Overview

Comet engaged OtterSec to perform an assessment of the `comet-contracts-v1` program. This assessment was conducted between January 2nd and January 15th, 2024. For more information on our auditing methodology, see Appendix B.

Key Findings

Over the course of this audit engagement, we produced 11 findings in total.

In particular, we identified a high-risk vulnerability regarding the failure to implement access control for the freeze status, allowing any user to change the freeze status of a pool (OS-CMT-ADV-01) and another issue concerning the absence of proper validation, which poses a risk of fund loss for users if the pool amount is inadvertently set to zero. In such cases, users may transfer tokens to the pool but receive no shares in return (OS-CMT-ADV-04).

Additionally, we discovered several cases of broken accounting from discrepancies in the math module due to a lack of precision and utilization of round-down division in math operations (OS-CMT-ADV-02) and also highlighted the erroneous transfer of the exit fee to the pool's address instead of transferring to factory contract (OS-CMT-ADV-05).

We also made recommendations regarding the importance of removing any unused or redundant code to enhance the readability and maintainability of the code base (OS-CMT-SUG-00) and suggested adherence to coding best practices and the necessity of a robust test suite (OS-CMT-SUG-01).

02 | Scope

The source code was delivered to us in a git repository at github.com/CometDEX/comet-contracts-v1. This audit was performed against commit 4abcb8c.

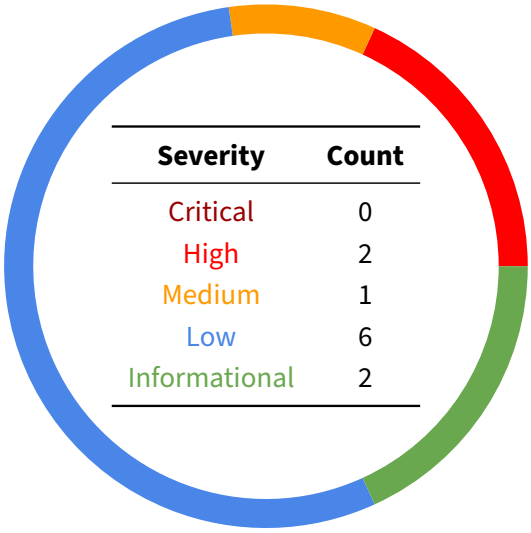
A brief description of the programs is as follows:

Name	Description
comet-contracts-v1	A decentralized automated market maker protocol, which is a fork of the Ethereum balancer protocol. It is built for the Soroban blockchain, separating the automated market maker curve logic and math from the core swapping functionality.

03 | Findings

Overall, we reported 11 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings do not have an immediate impact but will aid in mitigating future vulnerabilities.



Severity	Count
Critical	0
High	2
Medium	1
Low	6
Informational	2

04 | Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-CMT-ADV-00	High	Resolved	Invoking `execute_swap_exact_amount_out` may drain the pool.
OS-CMT-ADV-01	High	Resolved	`set_freeze_status` fails to implement any access control, allowing any user to change the freeze status of a pool.
OS-CMT-ADV-02	Medium	Resolved	Multiple cases of broken accounting due to discrepancies in the math module.
OS-CMT-ADV-03	Low	Resolved	`execute_unbind` fails to remove old records from `record_map` when a token is unbound.
OS-CMT-ADV-04	Low	Resolved	`join_pool` lacks proper validation, which loses funds for users if `pool_amount_out` is inadvertently set to zero, resulting in a situation where they transfer tokens to the pool but receive no shares in return.
OS-CMT-ADV-05	Low	Resolved	`execute_exit_pool` erroneously transfers the exit fee to the pool's address instead of transferring to `factory`.
OS-CMT-ADV-06	Low	Resolved	Burning liquidity pool tokens without appropriately adjusting the internal pool state may result in an inconsistency between the liquidity pool token supply and the actual liquidity.
OS-CMT-ADV-07	Low	Resolved	In `pool`, multiple instances exist where `EXIT_FEE` is utilized for the exit fees calculation instead of `exit_fee`.

OS-CMT-ADV-08	Low	Resolved	`execute_rebind` contains a redundant check on the number of bound tokens in the pool, which results in an error if the pool contains `MAX_BOUND_TOKENS`.

OS-CMT-ADV-00 [high] | Pool Drainage Due To Rounding Error

Description

In `execute_swap_exact_amount_out`, the pool may be drained as it does not enforce a minimum input amount (`token_amount_in`) greater than zero due to rounding errors that favor the user.

src/c_pool/call_logic/pool.rs RUST

```rust
pub fn execute_swap_exact_amount_out(
    [...]
) -> (i128, i128) {
    assert_with_error!(&e, !read_freeze(&e), Error::ErrFreezeOnlyWithdrawals);
    assert_with_error!(&e, token_amount_out >= 0, Error::ErrNegative);
    assert_with_error!(&e, max_amount_in >= 0, Error::ErrNegative);
    assert_with_error!(&e, max_price >= 0, Error::ErrNegative);
    assert_with_error!(&e, read_public_swap(&e), Error::ErrSwapNotPublic);
    [...]
}
```

Due to a lack of an explicit check for `token_amount_in` being greater than zero before updating the balances, if `token_amount_in` is zero or negative (due to rounding errors), the pool's input token balance (`in_record.balance`) would be increased by an insignificant amount. The output token balance (`out_record.balance`) would be reduced by `token_amount_out`. This may allow an attacker to drain the pool without providing a meaningful amount of input tokens.

Remediation

Check that `token_amount_in` is greater than zero before updating the balances.

src/c_pool/call_logic/pool.rs DIFF

```
diff --git a/contracts/src/c_pool/call_logic/pool.rs
    ↪  b/contracts/src/c_pool/call_logic/pool.rs
index f875b8c..7ef9a36 100644
--- a/contracts/src/c_pool/call_logic/pool.rs
+++ b/contracts/src/c_pool/call_logic/pool.rs
@@ -303,6 +303,7 @@ pub fn execute_swap_exact_amount_out(
         read_swap_fee(&e),
     );

+    assert_with_error!(&e, token_amount_in > 0, Error::ErrMathApprox);
     assert_with_error!(&e, token_amount_in <= max_amount_in, Error::ErrLimitIn);

     in_record.balance = c_add(&e, in_record.balance,
         ↪  token_amount_in).unwrap_optimized();
```

Patch

Fixed in 2e78fbf and dac8a38.

OS-CMT-ADV-01 [high]| Lack Of Access Control

Description

`set_freeze_status` in `comet` enables freezing or unfreezing of the pool. It is intended to be callable only by the pool admin. However, due to a lack of access control functionality, anyone may invoke `set_freeze_status`. As a result, malicious users may abuse `set_freeze_status` to disrupt the normal operation of the contract by randomly freezing the pool, affecting legitimate users of the pool.

src/c_pool/comet.rs RUST

```rust
// Only Callable by the Pool Admin
// Freezes Functions and only allows withdrawals
fn set_freeze_status(e: Env, caller: Address, val: bool) {
    execute_set_freeze_status(e, caller, val);
}
```

Remediation

Implement an access control mechanism such that `set_freeze_status` is only executable by the designated pool admin.

src/c_pool/comet.rs DIFF

```
diff --git a/contracts/src/c_pool/comet.rs b/contracts/src/c_pool/comet.rs
index 55cf4b6..c9acaa9 100644
--- a/contracts/src/c_pool/comet.rs
+++ b/contracts/src/c_pool/comet.rs
@@ -372,6 +372,8 @@ impl CometPoolTrait for CometPoolContract {
     // Only Callable by the Pool Admin
     // Freezes Functions and only allows withdrawals
     fn set_freeze_status(e: Env, caller: Address, val: bool) {
+        assert_with_error!(&e, caller == read_controller(&e),
    ↪  Error::ErrNotController);
+        caller.require_auth();
         execute_set_freeze_status(e, caller, val);
     }
```

Patch

Fixed in 808ab96.

OS-CMT-ADV-02 [med]| Inconsistencies In Math Module Implementation

Description

The precision of the fixed-point math in Comet is 10^7, which is low compared to that set in `BalancerV1`, which is 10^{18}. This low precision in the Comet system may be problematic in a scenario where there are two tokens with very different prices, one being significantly cheaper than the other, with the low precision (10^7), the representation of the prices may not have enough decimal places to accurately capture the differences between the high and low-priced tokens, rendering pools containing both high-priced and low-priced tokens unusable due to the lack of precision.

src/c_consts.rs RUST

```rust
//! Comet Pool Constants
use soroban_fixed_point_math::STROOP;

pub const BONE: i128 = STROOP as i128;
[...]
```

Furthermore, the operations for multiplications, divisions, and powers consistently round down, regardless of whether they favor the user or the pool. This lack of rounding consistency yields inaccuracies, especially in calculations involving pool invariants.

Remediation

Revert to the original BONE variable utilized in `BalancerV2`. Alternatively, a different split of bits for the decimal part may be implemented, such as $80/48$ bits, which would translate to a BONE value of 10^{14}. Additionally, adopt a reference implementation from BalancerV2 Weighted pools for the rounding issue, which correctly rounds every calculation against the user.

Patch

Fixed in dac8a38.

OS-CMT-ADV-03 [low] | Persistent Record Of Unbound Tokens

Description

In `execute_unbind`, within `bind`, when tokens are unbound, their old records are not removed from `record_map`, potentially resulting in a situation where the map's size becomes a limiting factor due to the ledger entry size limit. If the accumulated records for unbound tokens become substantial, they may become limited by the ledger's size constraints. The limitation on the map size may prevent the addition of new tokens to the liquidity pool when attempting to bind a new token,

src/c_pool/call_logic/bind.rs RUST

```rust
// Removes a specific token from the Liquidity Pool
pub fn execute_unbind(e: Env, token: Address, user: Address) {
    [...]
    record.balance = 0;
    record.bound = false;
    record.index = 0;
    record.denorm = 0;]
    record_map.set(last_token, record_current);
    record_map.set(token.clone(), record);
    write_record(&e, record_map);
    [...]
}
```

Remediation

Remove old records for tokens no longer part of the liquidity pool so that the `record_map` does not grow indefinitely and that space is available for new token bindings.

src/c_pool/call_logic/pool.rs DIFF

```diff
--- a/contracts/src/c_pool/call_logic/bind.rs
+++ b/contracts/src/c_pool/call_logic/bind.rs
@@ -144,13 +144,9 @@ pub fn execute_unbind(e: Env, token: Address, user: Address) {
     write_tokens(&e, tokens);
     let mut record_current =
         ↪  record_map.get(last_token.clone()).unwrap_optimized();
     record_current.index = index;
-    record.balance = 0;
-    record.bound = false;
-    record.index = 0;
-    record.denorm = 0;

     record_map.set(last_token, record_current);
-    record_map.set(token.clone(), record);
+    record_map.remove(token.clone());

     write_record(&e, record_map);
```

Patch

Fixed in e2fa992.

OS-CMT-ADV-04 [low] | Missing Parameter Validation

Description

Within `join_pool` in `pool`, a user may transfer tokens to the pool without receiving any shares in return when `pool_amount_out` is zero. When calling `join_pool` with `pool_amount_out` set to zero, instead of reverting, the function proceeds to execute the `execute_join_pool` logic, which calculates the ratio, iterates over tokens, and calculates the amount to deposit for each token. However, the calculated ratio is effectively zero Since `pool_amount_out` is zero. As a result, for each token, the calculated `token_amount_in` becomes zero (due to the ratio being zero).

Thus, users may transfer tokens to the pool without receiving any shares in return, which may be unintuitive and unexpected for users, as they might assume that providing tokens to the pool should always result in receiving a share of the pool.

Remediation

Include a specific check at the beginning of `join_pool` to ensure `pool_amount_out` is greater than zero. If `pool_amount_out` is zero, the function should revert with an appropriate error message.

src/c_pool/call_logic/pool.rs DIFF

```
diff --git a/contracts/src/c_pool/call_logic/pool.rs
    ↪  b/contracts/src/c_pool/call_logic/pool.rs
index f875b8c..02223de 100644
--- a/contracts/src/c_pool/call_logic/pool.rs
+++ b/contracts/src/c_pool/call_logic/pool.rs
@@ -45,7 +45,7 @@ pub fn execute_gulp(e: Env, t: Address) {

 pub fn execute_join_pool(e: Env, pool_amount_out: i128, max_amounts_in: Vec<i128>,
     ↪  user: Address) {
     assert_with_error!(&e, !read_freeze(&e), Error::ErrFreezeOnlyWithdrawals);
-    assert_with_error!(&e, pool_amount_out >= 0, Error::ErrNegative);
+    assert_with_error!(&e, pool_amount_out > 0, Error::ErrNegative);
     assert_with_error!(&e, read_finalize(&e), Error::ErrNotFinalized);
```

Patch

Fixed in 1c73981.

OS-CMT-ADV-05 [low] | Misallocation Of Exit Fee

Description

`execute_exit_pool` enables a user to exit the pool by supplying a specific quantity of pool shares. Following the computation of the exit fee, `execute_exit_pool` transfers this fee to the share contract identified by the address `share_contract_id`. Presently, this contract's address is determined using `e.current_contract_address()`, resulting in the pools address. Consequently, rather than directing the exit fee to the intended recipient, the `factory`, the amount is sent to the address of the pool.

c_pool/call_logic/pool.rs RUST

```rust
// Helps a user exit the pool
pub fn execute_exit_pool(e: Env, pool_amount_in: i128, min_amounts_out: Vec<i128>,
    ↪  user: Address) {
    [...]
    assert_with_error!(&e, ratio != 0, Error::ErrMathApprox);
    pull_shares(&e, user.clone(), pool_amount_in);

    let share_contract_id = e.current_contract_address();
    push_shares(&e, share_contract_id, EXIT_FEE);
    [...]
}
```

Remediation

Ensure the `exit_fee` is sent to the factory contract instead of to the pools address.

src/c_pool/call_logic/pool.rs DIFF

```
diff --git a/contracts/src/c_pool/call_logic/pool.rs
    ↪  b/contracts/src/c_pool/call_logic/pool.rs
index f875b8c..ae38413 100644
--- a/contracts/src/c_pool/call_logic/pool.rs
+++ b/contracts/src/c_pool/call_logic/pool.rs
@@ -113,8 +113,8 @@ pub fn execute_exit_pool(e: Env, pool_amount_in: i128,
    ↪  min_amounts_out: Vec<i128
     assert_with_error!(&e, ratio != 0, Error::ErrMathApprox);
     pull_shares(&e, user.clone(), pool_amount_in);

-    let share_contract_id = e.current_contract_address();
-    push_shares(&e, share_contract_id, EXIT_FEE);
+    let factory = read_factory(&e);
+    push_shares(&e, factory, EXIT_FEE);
     burn_shares(&e, pai_after_exit_fee);
     let tokens = read_tokens(&e);
     let mut records = read_record(&e);
```

Patch

Fixed in bb5c006.

OS-CMT-ADV-06 [low] | Incorrect Implementation Of Burn Logic

Description

While burning liquidity pool tokens, in its current implementation, `burn` burns tokens (`amount`) from an account (`from`), but it fails to adjust the number of tokens in that liquidity pool.

src/c_pool/comet.rs RUST

```rust
fn burn(e: Env, from: Address, amount: i128) {
    from.require_auth();
    check_nonnegative_amount(amount);
    e.storage()
        .instance()
        .extend_ttl(SHARED_LIFETIME_THRESHOLD, SHARED_BUMP_AMOUNT);
    spend_balance(&e, from.clone(), amount);
    TokenUtils::new(&e).events().burn(from, amount);
}
```

Thus, when a liquidity provider calls `burn` to withdraw their liquidity and claim the fees, the contract only burns the tokens from the liquidity provider's account. Still, it does not adjust the pool's internal state (`total_shares`), resulting in an inconsistency where the liquidity pool token supply suggests the presence of more liquidity than what is contained in the pool. This disparity in the amount of liquidity pool tokens may impact the decentralized exchange's pricing mechanism and overall stability.

Remediation

Ensure when invoking `burn`, it also adjusts the relevant internal state of the liquidity pool by decrementing `total_shares`.

Patch

Fixed in d165f1e.

OS-CMT-ADV-07 [low] | Error In Exit Fee Calculation

Description

In `pool`, `execute_exit_pool` utilizes `push_shares`, passing the `EXIT_FEE` parameter for exit fee calculation. Similarly, in `execute_wdr_tokn_amt_in_get_lp_tokns_out` and `execute_wdr_tokn_amt_out_get_lp_tokns_in`, `push_shares` and `burn_shares` are invoked with the `EXIT_FEE` parameter. This results in an inaccurate exit fee calculation, as the appropriate variable to use is `exit_fee`, not `EXIT_FEE`.

pair/src/lib.rs RUST

```rust
pub fn execute_exit_pool(e: Env, pool_amount_in: i128, min_amounts_out: Vec<i128>,
    ↪  user: Address) {
    [...]
    push_shares(&e, share_contract_id, EXIT_FEE);
    [...]
}

pub fn execute_wdr_tokn_amt_in_get_lp_tokns_out(
    e: Env,
    token_out: Address,
    pool_amount_in: i128,
    min_amount_out: i128,
    user: Address,
) -> i128 {
    [...]
    pull_shares(&e, user.clone(), pool_amount_in);
    burn_shares(&e, c_sub(&e, pool_amount_in, EXIT_FEE).unwrap_optimized());
    let factory = read_factory(&e);
    push_shares(&e, factory, EXIT_FEE);
    [...]
}
```

The disparity in handling exit fees could result in unpredictable behavior, potentially charging users an incorrect amount upon exiting the pool. Such inconsistencies may impact the liquidity pool's economic incentives and overall stability. Additionally, within `execute_rebind`, a hardcoded value of zero is employed instead of utilizing `EXIT_FEE`, resulting in the exit fee being calculated by multiplying `token_balance_withdrawn` by zero. Consequently, the value of `token_exit_fee` will consistently be zero, irrespective of the actual withdrawal amount or the configured exit fee.

Remediation

Utilize `exit_fee` to derive the exit fees when pushing or burning shares in `execute_exit_pool`, `execute_wdr_tokn_amt_in_get_lp_tokns_out` and `execute_wdr_tokn_amt_out_get_lp_tokns_in`. Additionally, utilize `EXIT_FEE` in `execute_rebind` instead of a hard-coded value of zero for calculating `token_exit_fee`.

Patch

Fixed in 9de9f7c.

OS-CMT-ADV-08 [low] | Faulty Token Limit Check

Description

In `execute_rebind` within `bind`, an inaccurate validation exists concerning the limit on the number of bound tokens in the pool. The assert statement, which examines whether `read_tokens(&e).len() < MAX_BOUND_TOKENS`, mistakenly rejects valid rebind operations when adjusting the values of an already bound token. The maximum number of bound tokens is already verified in `execute_bind`. Therefore, when calling `execute_rebind`, if the pool already contains the maximum allowable tokens (i.e., `MAX_BOUND_TOKENS`, which is set to eight tokens), this assert statement will fail, hindering the addition of further tokens, even if they are within the allowable limit.

pair/src/lib.rs RUST

```rust
pub fn execute_rebind(e: Env, token: Address, balance: i128, denorm: i128, admin:
    ↪  Address) {
    assert_with_error!(&e, balance >= 0, Error::ErrNegative);
    assert_with_error!(&e, !read_finalize(&e), Error::ErrFinalized);
    assert_with_error!(
        &e,
        read_tokens(&e).len() < MAX_BOUND_TOKENS,
        Error::ErrMaxTokens
    );
    [...]
}
```

Remediation

Exclude this assertion from `execute_rebind`, as the validation performed in `execute_bind` is sufficient to prevent the inclusion of additional tokens beyond the limit specified in `MAX_BOUND_TOKENS`.

pair/src/lib.rs DIFF

```
diff --git a/contracts/src/c_pool/call_logic/bind.rs
    ↪  b/contracts/src/c_pool/call_logic/bind.rs
index 8316010..9068214 100644
--- a/contracts/src/c_pool/call_logic/bind.rs
+++ b/contracts/src/c_pool/call_logic/bind.rs
@@ -51,11 +51,6 @@ pub fn execute_rebind(e: Env, token: Address, balance: i128,
    ↪  denorm: i128, admin
     assert_with_error!(&e, balance >= 0, Error::ErrNegative);
     assert_with_error!(&e, !read_finalize(&e), Error::ErrFinalized);

-    assert_with_error!(
-        &e,
-        read_tokens(&e).len() < MAX_BOUND_TOKENS,
-        Error::ErrMaxTokens
-    );
```

Patch

Fixed in 6ffd7de.

05 | General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent anti-patterns and may result in security issues in the future.

ID	Description
OS-CMT-SUG-00	Recommendations to remove unused or redundant code to improve the readability and maintainability of the code base.
OS-CMT-SUG-01	Suggestions regarding adherence to coding best practices and a robust test suite.

OS-CMT-SUG-00 | Removal Of Unused / Redundant code

Description

1. The code base contains several instances of unused or redundant code, including:
   (a) Calling `read_tokens` twice in `execute_bind`.

c_pool/call_logic/bind.rs RUST

```
pub fn execute_bind(e: Env, token: Address, balance: i128, denorm: i128,
    ↪  admin: Address) {
    [...]
    let index = read_tokens(&e).len();
    assert_with_error!(&e, index < MAX_BOUND_TOKENS,
        ↪  Error::ErrMaxTokens);
    let mut tokens_arr = read_tokens(&e);
    let mut record_map = read_record(&e);
    [...]
}
```

   (b) `token_amount_in` will never be zero in `execute_join_pool`, thus the check below is redundant:

c_pool/call_logic/pool.rs RUST

```
pub fn execute_join_pool(e: Env, pool_amount_out: i128, max_amounts_in:
    ↪  Vec<i128>, user: Address) {
   [...]
        let token_amount_in =
            c_add(&e, c_mul(&e, ratio, rec.balance).unwrap_optimized(),
                ↪  1).unwrap_optimized();
        if token_amount_in == 0 {
            panic_with_error!(&e, Error::ErrMathApprox);
        }
}
```

   (c) The code contains unused storage types which should be removed:

src/c_pool/storage_types.rs RUST

```
pub enum DataKeyToken {
    [...]
    Nonce(Address),
    State(Address),
    Admin,
}
```

2. Ensure the removal of unused variables and function arguments in the code base in adherence to coding best practices and overall maintainability and readability. Some such examples are:
   (a) The `caller` argument in the setter functions.
   (b) The `val` variable in `execute_init`.
   (c) The `contract_address` variable in `mint_shares`.

Remediation

Ensure the above-mentioned changes are implemented.

Patch

Fixed in dac8a38.

OS-CMT-SUG-01 | Code Maturity

Description

1. The absence of emitted events in administrative tasks such as modifying swap fee, public swap, controller, or freeze state variables in `setter` and during minting of shares in `token_utility`, within `mint_shares` indicates that there are no events triggered or logged when these changes occur. Events are crucial for providing transparency and allowing external systems or users to track the state changes in a contract.
2. In `metadata`, within `read_swap_fee`, if `SwapFee` is not discovered in the storage, it returns zero as the default value. Instead of returning zero, returning `MIN_FEE` is more suitable to improve clarity. It should be noted that this is an unreachable case.

src/c_pool/metadata.rs RUST

```rust
pub fn read_swap_fee(e: &Env) -> i128 {
    let key = DataKey::SwapFee;
    e.storage()
        .instance()
        .get::<DataKey, i128>(&key)
        .unwrap_or(0)
}
```

3. The code base should be enhanced with more comprehensive tests encompassing all possible scenarios, including instances such as a pool containing assets with significant price differences.

Remediation

1. Ensure the following function emits functions on state change:
    - `execute_set_swap_fee`.
    - `execute_set_controller`.
    - `execute_set_public_swap`.
    - `execute_set_freeze_status`.
    - `mint_shares`.
2. Replace the literal `0` with `MIN_FEE` in `read_swap_fee`.
3. Ensure the inclusion of additional tests to make the code more robust.

Patch

Fixed in dac8a38.

A | Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings may be found in the General Findings section.

Critical	Vulnerabilities that immediately result in a loss of user funds with minimal preconditions. Examples: • Misconfigured authority or access control validation. • Improperly designed economic incentives leading to loss of funds.
High	Vulnerabilities that may result in a loss of user funds but are potentially difficult to exploit. Examples: • Loss of funds requiring specific victim interactions. • Exploitation involving high capital requirement with respect to payout.
Medium	Vulnerabilities that may result in denial of service scenarios or degraded usability. Examples: • Computational limit exhaustion through malicious input. • Forced exceptions in the normal user flow.
Low	Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk. Examples: • Oracle manipulation with large capital requirements and multiple transactions.
Informational	Best practices to mitigate future security risks. These are classified as general findings. Examples: • Explicit assertion of critical internal invariants. • Improved input validation.

B | Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the program's implementation requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to comprehensively understand the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that the other missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.