Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

WOMBAT

Wombat-Exchange

Veridise Inc.

Dec. 24 2024

► **Prepared For:**

Wombat Finance Limited
https://www.wombat.exchange

► **Prepared By:**

Alberto Gonzalez
Ajinkya Rajput

► **Contact Us:**

contact@veridise.com

► **Version History:**

Dec. 24, 2024	V2
Nov. 12, 2024	V1
Oct. 22, 2024	Initial Draft

Contents

1 Executive Summary

From Sep. 2, 2024 to Oct. 29, 2024, Wombat Finance Limited engaged Veridise to review the security of their Wombat-Exchange protocol. Wombat-Exchange is a decentralized exchange (DEX) platform. The project includes an (AMM) system with a unique curve invariant designed for stable-swaps based on an asset-liability model which uses the coverage ratio of pools instead of tokens' reserves.

Veridise conducted the assessment over 17 person-weekss, with 2 engineers reviewing code over 8.5 weekss on commit `fbea044`. The auditing strategy involved extensive manual code review performed by Veridise engineers.

Project summary. As mentioned above, Wombat-Exchange is a decentralized exchange (DEX) platform designed to facilitate efficient, stable-swaps through an innovative Automated Market Maker model. It also features a reward mechanism that provides incentives for liquidity providers using WOM tokens and other third-party tokens. Additionally, the platform incorporates a governance model with veTokens to handle voting as well as to boost users liquidity provision rewards.

The protocol is structured into the following key contracts:

- `PoolV3:` This contract implements the AMM logic within the Wombat-Exchange ecosystem. It calculates the amounts for deposits, withdrawals, and swaps, and requests the necessary cash and liability updates in the `Asset` contract. The `HighCovRatioFeePoolV3` and `DynamicPoolV3` are variants of `PoolV3`, modified to handle riskier stable tokens as well as liquid staking tokens.
- `Asset:` This contract represents the LP tokens within the AMM. It tracks the cash and liability balances necessary for the asset-liability model, enabling single-sided liquidity provision. The `DynamicAsset` and `PriceFeedAsset` are variant of the `Asset` contract, modified to handle the price change of liquid staking tokens in relation to the staked token (e.g. stETH and ETH).
- `BoostedMasterWombat:` This contract rewards users with `Wom` tokens based on their staked LP tokens, allowing rewards boosting through the users' `veWom` balance.
- `BoostedMultiRewarder:` This contract is equivalent to the `BoostedMasterWombat` contract but allows rewarding users with different tokens in addition to the `Wom` token.
- `VeWom:` This contract manages the locking of `Wom` tokens, allowing users to mint `veWom` tokens for governance voting power and to boost their staking positions for additional rewards.

Code Assessment. The Wombat-Exchange developers provided the source code of the protocol for review. The source code is original code written by the Wombat-Exchange developers intended to replicate their existing Solidity codebase into a Rust one for the Soroban runtime.

The codebase included several inline comments, which provide some context for the logic implemented within the contracts. In addition, the auditors were provided with previous audit

reports and whitepapers*, which supplemented the understanding of the protocol's design and objectives.

The codebase included a test suite for most of the contracts within the scope of the audit. However, the auditors noted that the test scenarios mainly focused on straightforward, expected use cases. There is an opportunity to improve the test suite by incorporating simulations of complex scenarios and adversarial conditions, such as unauthorized access to sensitive functionality or a variety of reward distribution scenarios. Implementing these tests could have helped identify some of the issues highlighted in this report such as: V-WOM-VUL-001, V-WOM-VUL-003, V-WOM-VUL-004, V-WOM-VUL-006 and V-WOM-VUL-013.

Summary of Issues Detected. The review uncovered 34 issues, with 7 assessed as high or critical severity by the Veridise team. Specifically, the analysts identified that the access-control logic in the `BoostedMultiRewarder` contract's upgrade functionality can be bypassed by any user, leading to unauthorized upgrades (V-WOM-VUL-001). They also noted an incorrect implementation in the computation of the new equilibrium coverage ratio (V-WOM-VUL-002). Furthermore, due to an error in calculating users' pending rewards, any user can drain funds from the reward contracts (V-WOM-VUL-003). Additionally, the lack of access control in the price reporting functionality of the `GovernedPriceFeed` contract allows malicious users to report incorrect asset prices (V-WOM-VUL-004) and the lack of access control in the update factor functionality of the `BoostedMasterWombat` contract allows manipulation of boosted rewards (V-WOM-VUL-006). Lastly, improper use of the Soroban instance storage in the `veWOM` contract could lead to a denial-of-service state (V-WOM-VUL-007).

The Veridise team also identified 7 medium-severity issues. Notably, they found that a malicious user could disrupt the deposit functionality of any newly added token in the pool (V-WOM-VUL-008). Additionally, the `BoostedMultiRewarder` contract fails to save changes to the rewards configuration in storage (V-WOM-VUL-009). The team highlighted incorrect implementations of functions for division operations and swap quote calculations (V-WOM-VUL-010, V-WOM-VUL-011). A missing check for unpaid rewards in the `MultiRewarderPerSec` contract also results in the loss of rewards for users (V-WOM-VUL-013).

Additionally, the Veridise analysts identified 16 low-severity, 2 warning, and 2 informational findings. Some of these findings include the absence of explicit underflow checks in cash and liability updates within the `Asset` contract (V-WOM-VUL-016) and the use of a stale exchange rate during calculations between liability and liquidity tokens in the pool contracts due to unconsidered pending fees (V-WOM-VUL-019). Other concerns were that adding new rewards during an active distribution period in the `BoostedMultiRewarder` contract could cause the loss of rightful rewards for current stakers (V-WOM-VUL-020), the relaxed access control in the `BoostedMultiRewarder` contract's upgrade functionality (V-WOM-VUL-022), and that the slippage protection in the `withdraw_from_other_asset` function fails if the operation is not executed atomically (V-WOM-VUL-025).

Recommendations. After conducting the assessment of the protocol, the security analysts had a few suggestions to improve the Wombat-Exchange protocol security before deployment:

* The whitepapers can be found on Wombat's website at `https://docs.wombat.exchange/resources/whitepapers`

Centralization Risk. Due to the privileged access granted to the owner of Wombat-Exchange contracts, which includes the authority to upgrade contract implementations, the auditors recommend implementing robust key management practices and secure signing procedures.

Typecasts. The codebase contains numerous instances of unsafe typecasting, where values are cast using rust expressions like "as u128" and "as i128" without verifying that the values fall within the acceptable range of the target type. This practice can lead to unexpected behavior or runtime errors if the values exceed the bounds of the target type, potentially resulting in overflow issues. In contrast, the Solidity codebase includes functions for safe typecasting, which ensure that values are within the permissible range before conversion. It is recommended to adopt similar safe typecasting methods in the Soroban codebase.

Refactoring. The codebase contains several instances of duplicated logic across different files, which introduces maintenance challenges and risks of inconsistency. Refactoring the code to consolidate these repeated sections into shared modules or libraries is recommended. By centralizing common functionality, the codebase will become more maintainable, reducing the likelihood of errors and ensuring that updates or bug fixes are consistently applied across the system. Examples include:

- **Mathematical Utilities**: The `dsmath.rs` and `signed_safe_math.rs` files, which implement functions such as *sqrt*, *wmul*, and *wdiv*, are repeated across several modules in the codebase. This redundancy led to a subtle issue, where one instance of the *wdiv* function was incorrectly implemented (V-WOM-VUL-010), while other instances were correctly implemented.
- **Pool Logic**: The codebase includes three pool types `PoolV3`, `HighCovRatioFeePoolV3`, and `DynamicPool` which share most of their logic from the files `core_v3.rs` and `pool_v3_logic.rs`. However, each pool maintains its own instance of these files, creating code duplication that requires identical fixes across different files, as highlighted in issues like V-WOM-VUL-002, V-WOM-VUL-008, V-WOM-VUL-011, V-WOM-VUL-016, V-WOM-VUL-017, V-WOM-VUL-019, and V-WOM-VUL-025.
- **Ownership Management**: The `ownable_contract.rs` files across different modules (e.g., `BoostedMasterWombat`, `Asset`, `PoolV3`) contain identical functions for managing ownership, such as *check_owner*, *read_owner*, *write_ownership*, *initialize_owner*, and *has_owner*. These functions could be centralized into a single module that can be imported wherever ownership management is needed.

Differential Testing. Given that the Soroban version of the codebase is a direct translation of an already deployed and battle-tested Solidity version, it is crucial to ensure that both implementations exhibit equivalent behavior. Implementing differential testing can serve as an effective strategy to achieve this. By comparing the outputs and behaviors of the Soroban and Solidity versions under identical conditions, differential testing can help identify discrepancies and ensure functional parity. This approach leverages the robustness of the existing Solidity implementation to validate the new Soroban code, thereby enhancing confidence in its reliability and correctness before deployment. Moreover, differential testing would have been instrumental in catching issues such as V-WOM-VUL-002, V-WOM-VUL-003, V-WOM-VUL-006, V-WOM-VUL-011 and V-WOM-VUL-013.

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
Wombat-Exchange	`fbea044`	Rust	Soroban

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Sep. 2–Oct. 29, 2024	Manual	2	17 person-weekss

Table 2.3: Vulnerability Summary.

Name	Number	Acknowledged	Fixed
Critical-Severity Issues	4	4	4
High-Severity Issues	3	3	2
Medium-Severity Issues	7	7	4
Low-Severity Issues	16	16	6
Warning-Severity Issues	2	2	2
Informational-Severity Issues	2	2	2
TOTAL	34	34	20

Table 2.4: Category Breakdown.

Name	Number
Logic Error	14
Data Validation	5
Access Control	4
Maintainability	4
Denial of Service	3
Authorization	2
Overflow	1
Usability Issue	1

3 Security Assessment Goals and Scope

3.1 Security Assessment Goals

The engagement was scoped to provide a security assessment of Wombat-Exchange's smart contracts. During the assessment, the security analysts aimed to answer questions such as:

- Are access control mechanisms, such as ownership and role-based permissions, implemented and applied correctly across all contracts?
- Are the contract's initialization functions implemented correctly?
- Are the contracts vulnerable to common issues in Soroban projects, such as incorrect management of storage?
- Are there any discrepancies in logic between the Soroban and Solidity implementations that could affect the security of the project?
- Are standard AMM invariants –such as path independence, liquidity symmetry, and liquidity sensitivity– maintained?
- Are the AMM's users appropriately protected from sandwich attacks?
- Are the mathematical operations, particularly those involving swaps, liquidity computations, fees and reward calculations, implemented correctly?
- Are the mechanisms for staking LP tokens and distributing rewards secure and fair, preventing any form of manipulation?
- Can users prevent other users from receiving their staking rewards?
- Is the Wom locking mechanism secure, ensuring that lock periods and veWom computations are handled correctly?

3.2 Security Assessment Methodology & Scope

Security Assessment Methodology. To address the questions above, the security assessment involved a combination of human experts and extensive manual review.

Scope. The scope of this security assessment was limited to the following files:

- `contracts/1_1_ERC20/*`
- `contracts/1_3_Asset/*`
- `contracts/2_1_PoolV3/*`
- `contracts/2_2_HighCovRatioFeePoolV3/*`
- `contracts/3_1_BoostedMultiRewarder/*`
- `contracts/3_2_BoostedMasterWombat/*`
- `contracts/4_4_VeWom/*`
- `contracts/CoreV3/*`
- `contracts/DSMath/*`
- `contracts/SignedSafeMath/*`
- `contracts/1_4_DynamicAsset/*`

- `contracts/1_5_GovernedPriceFeed/*`
- `contracts/1_6_PriceFeedAsset/*`
- `contracts/2_3_DynamicPoolV3/*`
- `contracts/4_1_MultiRewarderPerSec/*`

Methodology. Veridise security analysts reviewed the reports of previous audits for the Wombat-Exchange inspected the provided tests, and read the Wombat-Exchange documentation. They then began a manual review of the code. During the review, the Veridise auditors maintained active communication with the Wombat-Exchange developers to ask questions about the code.

3.3 Classification of Vulnerabilities

When Veridise security analysts discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise.

The severity of a vulnerability is evaluated according to the Table 3.1.

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

The likelihood of a vulnerability is evaluated according to the Table 3.2.

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

The impact of a vulnerability is evaluated according to the Table 3.3:

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4 Vulnerability Report

This section presents the vulnerabilities found during the security assessment. For each issue found, the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.) is specified. Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-WOM-VUL-001	Unauthorized Contract Upgrade Possible . . .	Critical	Fixed
V-WOM-VUL-002	Incorrect Implementation of New . . .	Critical	Fixed
V-WOM-VUL-003	Incorrect reward calculation allows users . . .	Critical	Fixed
V-WOM-VUL-004	Missing authorization check in . . .	Critical	Fixed
V-WOM-VUL-005	Token can be re-initialized	High	Fixed
V-WOM-VUL-006	Unauthorized Update of User Factors in . . .	High	Fixed
V-WOM-VUL-007	Unbounded User Data in Instance Storage . . .	High	Partially Fixed
V-WOM-VUL-008	Deposit functionality can be permanently . . .	Medium	Acknowledged
V-WOM-VUL-009	Changes to reward_info are not saved in . . .	Medium	Fixed
V-WOM-VUL-010	Incorrect Denominator in wdiv()	Medium	Fixed
V-WOM-VUL-011	Incorrect implementation of . . .	Medium	Fixed
V-WOM-VUL-012	Temporary storage usage for critical data . . .	Medium	Acknowledged
V-WOM-VUL-013	Loss of unpaid rewards	Medium	Fixed
V-WOM-VUL-014	Incorrect haircut calculation due to scale . . .	Medium	Acknowledged
V-WOM-VUL-015	Centralization Risk	Low	Acknowledged
V-WOM-VUL-016	Potential Underflow in Cash and Liability . . .	Low	Fixed
V-WOM-VUL-017	Potential underflow in . . .	Low	Fixed
V-WOM-VUL-018	Integer Square Root Functions Return . . .	Low	Acknowledged
V-WOM-VUL-019	Exchange rate might be stale in pool . . .	Low	Partially Fixed
V-WOM-VUL-020	Incorrect reward calculation in . . .	Low	Acknowledged
V-WOM-VUL-021	Unsafe typecasts in the codebase	Low	Acknowledged
V-WOM-VUL-022	Multi rewarder upgrade should only . . .	Low	Fixed
V-WOM-VUL-023	Data inconsistency due to outdated data array	Low	Fixed
V-WOM-VUL-024	Unauthorized access to return functions . . .	Low	Fixed
V-WOM-VUL-025	Potential loss due to non-atomic . . .	Low	Acknowledged
V-WOM-VUL-026	Inconsistent Tip Bucket Balance Due to . . .	Low	Acknowledged
V-WOM-VUL-027	During a swap, the toAsset paused is not . . .	Low	Fixed
V-WOM-VUL-028	fill_pool may put the protocol out of . . .	Low	Acknowledged
V-WOM-VUL-029	Stale price data in GovernedPriceFeed	Low	Acknowledged
V-WOM-VUL-030	Incorrect cash check due to scale factor . . .	Low	Acknowledged
V-WOM-VUL-031	No way to change or renounce ownership . . .	Warning	Fixed
V-WOM-VUL-032	User Can Keep Exceeding the Maximum . . .	Warning	Fixed
V-WOM-VUL-033	Inconsistent lock days validation	Info	Fixed
V-WOM-VUL-034	Typos, redudant code and other minor issues	Info	Fixed

4.1 Detailed Description of Issues

4.1.1 V-WOM-VUL-001: Unauthorized Contract Upgrade Possible Due to Insufficient Access Control

Severity	Critical	**Commit**	fbea044
Type	Access Control	**Status**	Fixed
File(s)	`boosted_multi_rewarderd_logic.rs`		
Location(s)	upgrade()		
Confirmed Fix At	`https://github.com/wombat-exchange/wombat-stellar-contract/commit/c911146df84224b96afdfc1fd4decdf93cc4a173`, c911146		

The `upgrade()` function in the `BoostedMultiRewarder` contract allows for upgrading the contract's WASM code. However, the current implementation has a flaw in its access control mechanism, allowing unauthorized users to perform contract upgrades.

The function uses two methods for access control:

1. `admin.require_auth()`: This ensures that the `admin` has authorized the transaction. Note that the argument named `admin` is an arbitrary address provided by the caller.
2. `has_operator_role(e, admin)`: This checks if the `admin` has the operator role.

```
pub fn upgrade(e: &Env, admin: Address, new_wasm_hash: BytesN<32>) {
    admin.require_auth();
    has_operator_role(e, admin);

    e.deployer().update_current_contract_wasm(new_wasm_hash);
}
```

Snippet 4.1: `upgrade` function from the `boosted_multi_rewarder_logic.rs` file.

The issue lies in the second check. The `has_operator_role()` function returns a boolean value, but the `upgrade()` function doesn't use this return value to revert the transaction if the `admin` lacks the operator role. As a result, any user can upgrade the contract, regardless of whether they have the operator role or not.

Impact This issue allows unauthorized contract upgrades.

Recommendation To fix this issue, the `upgrade()` function should explicitly check the return value of `has_operator_role()` and revert the transaction if it returns `false`.

Developer Response The developers implemented the suggested fix.

4.1.2 V-WOM-VUL-002: Incorrect Implementation of New Equilibrium Coverage Ratio Calculation

Severity	Critical	**Commit**	fbea044
Type	Logic Error	**Status**	Fixed
File(s)	`PoolV3/core_v3.rs`, `HighCovRatioFeePoolV3/core_v3.rs`, `DynamicPoolV3/core_v3.rs`, `CoreV3/lib.rs`		
Location(s)	_new_equil_cov_ratio()		
Confirmed Fix At	https://github.com/wombat-exchange/wombat-stellar-contract/commit/83c6737f30909c403b5e628cec942bf22163d06b, 86bba73		

The `_new_equil_cov_ratio()` function is meant to implement equation (6) from the Wombat whitepaper, which calculates the new equilibrium coverage ratio after a deposit or withdrawal. However, the current implementation is incorrect, as it fails to add `delta_i` to the numerator of the calculation.

The correct implementation should be:

- `i_wdiv(delta_i + i_wmul(sl, er), delta_i + sl)`

Instead, the current implementation is:

- `delta_i + i_wdiv(i_wmul(sl, er), delta_i + sl)`

Impact This error affects all operations that rely on the new equilibrium coverage ratio, including deposits and withdrawals in pools where r* != 1.

Recommendation The `_new_equil_cov_ratio()` function should be corrected to properly implement equation (6) from the whitepaper:

```
fn _new_equil_cov_ratio(er: i128, sl: i128, delta_i: i128) -> i128 {
    i_wdiv(delta_i + i_wmul(sl, er), delta_i + sl)
}
```

Snippet 4.2: Suggested fix for `_new_equil_cov_ratio`.

Developer Response The developers implemented the suggested fix in all the identified instances.

4.1.3 V-WOM-VUL-003: Incorrect reward calculation allows users to drain WOM balance

Severity	Critical	Commit	fbea044
Type	Logic Error	Status	Fixed
File(s)	boosted_master_wombat_logic.rs, boosted_multi_rewarder_logic.rs		
Location(s)	update_factor(), on_update_factor()		
Confirmed Fix At	https://github.com/wombat-exchange/wombat-stellar-contract/commit/04474bf3c3680fcddc80ff80791453275d89f54b, 04474bf		

The `update_factor()` function in the `BoostedMasterWombat` contract is responsible for updating a user's factor (which affects their reward multiplier) when their `veWom` balance changes. However, there is a flaw in the reward calculation logic that can lead to users receiving significantly more rewards than intended, allowing them to drain the contract's `wom` balance.

The issue stems from the calculation of `pending` rewards in the `update_factor()` and `on_update_factor()` functions. This computation determines how many `wom` tokens a user has earned since their last interaction with the contract. However, the current implementation forgets to subtract the user's `reward_debt`. The relevant code snippet is as follows:

```
let pending = _get_reward_debt(
                                user.amount,
                                current_pool.acc_wom_per_share,
                                user.factor,
                                current_pool.acc_wom_per_factor_share,
                            );

// increase pendingWom
user.pending_wom += pending
```

Snippet 4.3: Code snippet from the `update_factor()` function of the `boosted_master_wombat_logic.rs` contract.

This incorrect calculation leads to an inflation of rewards each time `update_factor()` is called, as it adds the total accumulated rewards to `pending_wom` without subtracting `reward_debt`.

It is important to note that the same issue appears in the `on_update_factor()` function of the `boosted_multi_rewarder_contract` contract:

```
if user_reward.reward_debt > 0 {
     user_reward.unpaid_rewards += _get_reward_debt(
         user_balance.amount,
         reward_info.acc_token_per_share,
         user_balance.factor,
         reward_info.acc_token_per_factor_share,
     );
 }
```

Snippet 4.4: Code snippet from `on_update_factor()` in `boosted_multi_rewarder_logic.rs`.

Impact This issue allows users to artificially inflate their rewards by triggering frequent updates to their `veWom` balance. Therefore, a malicious user could drain the entire `wom` balance of the contract.

Recommendation To fix this issue, the `update_factor()` and `on_update_factor()` functions should subtract the user's `reward_debt` from `pending`.

Developer Response The developers implemented the suggested fix.

4.1.4 V-WOM-VUL-004: Missing authorization check in set_latest_price() function

Severity	Critical	**Commit**	fbea044
Type	Access Control	**Status**	Fixed
File(s)	`GovernedPriceFeed/logic.rs`		
Location(s)	set_latest_price()		
Confirmed Fix At	`https://github.com/wombat-exchange/wombat-stellar-contract/commit/742e6f3a0b2c99fdf0106880fa8edd05a8df8dc4`, 742e6f3		

The `set_latest_price()` function in the `GovernedPriceFeed` contract is responsible for updating the price of a token. The function uses `has_operator_role()` to verify if the `sender` has the necessary operator role to perform this action. However, it lacks a crucial authorization check to ensure that the `sender` has explicitly authorized the transaction. This missing check will lead to unauthorized updates to the price, as any malicious user can set `sender` parameter to an account that do have the operator role.

Impact The absence of the authorization check in the `set_latest_price()` function will lead to unauthorized users updating the price of a token.

Recommendation To mitigate this issue, it is recommended to implement an explicit authorization check using `require_auth()` for the `sender` within the `set_latest_price()` function. As another option, consider using the function `check_access_control()` which validates both that `sender` has the operator role and that `sender` has authorized the contract call.

Developer Response The developers implemented the suggested fix.

4.1.5 V-WOM-VUL-005: Token can be re-initialized

Severity	High	**Commit**	fbea044
Type	Maintainability	**Status**	Fixed
File(s)	`ERC20/erc20_contract.rs`		
Location(s)	initialize()		
Confirmed Fix At	`https://github.com/wombat-exchange/wombat-stellar-contract/commit/033d43e3096aff0bd65e0d0031784f88423e08e5`, 033d43e		

The `Token` contract in `1_1_ERC20/src/erc20_contract.rs` file implements an ERC20-like token on the Soroban platform. It includes an `initialize()` function that sets up the token's metadata, including its decimal places, name, and symbol. However, this function lacks a check to prevent multiple initializations.

This function can be called multiple times without any restrictions, allowing anyone to change the token's properties after its initial setup.

Impact The token's core properties (decimal places, name, and symbol) can be altered at any time. An attacker can exploit this issue by reducing the number of decimals for any given tokens and potentially allowing the attacker to drain funds from contracts that make use of these tokens.

Recommendation Implement a one-time initialization check to ensure the `initialize()` function can only be called once.

Developer Response The developers implemented the suggested fix.

4.1.6 V-WOM-VUL-006: Unauthorized Update of User Factors in veWOM Balance Changes

Severity	High	**Commit**	fbea044
Type	Access Control	**Status**	Fixed
File(s)	`boosted_master_wombat_logic.rs`		
Location(s)	update_factor()		
Confirmed Fix At	`https://github.com/wombat-exchange/wombat-stellar-contract/commit/efaf851f90c3ef4f831be018c4001e4a92cf5960`, efaf851		

The `update_factor()` function in the `BoostedMasterWombat` contract is designed to update user factors when their `veWom` balance changes. This function plays a role in the contract's reward distribution mechanism, as it affects the calculation of user rewards across multiple pools. However, the current implementation lacks proper access control, allowing any user to call this function and manipulate the reward distribution system.

Impact Loss of funds for honest users who will receive fewer rewards than they should due to a malicious user increasing its factor unfairly.

Recommendation Only allow the `veWom` contract to call `update_factor()`.

Developer Response The developers implemented the suggested fix.

4.1.7 V-WOM-VUL-007: Unbounded User Data in Instance Storage Leads to Denial of Service

Severity	High	**Commit**	fbea044
Type	Denial of Service	**Status**	Partially Fixed
File(s)	`vewom_storage.rs`, `boosted_master_wombat_storage_types.rs`		
Location(s)	See description		
Confirmed Fix At	`https://github.com/wombat-exchange/wombat-stellar-contract/commit/b70d2fcd7e4d5ec663d80d07126f23407e3a9d3f`, b70d2fc		

The `VeWom` contract and the `BoostedMasterWombat` contract both store some information in instance storage, which has limited capacity. However, some of this saved data is unbounded, which can lead the contracts into a denial of service state given the limited capacity of instance storage.

- `VeWom` contract.
  - The contract uses the `DataKeyUserInfo` enum to store breeding information for each user and this enum is implemented with the `DataKeyTrait`, which uses instance storage for data persistence. As the number of users locking their `wom` tokens grows, the storage requirements will increase linearly.
  - Moreover, the contract also declares the enum `DataKeyVeWom` which also uses instance storage. Inside this enum, the `UsedVote(Address)` key is also unbounded.
- `BoostedMasterWombat` contract. The `BoostedMasterWombatDataKey` enum also uses instance storage for certain keys:
  - `AssetPid(Address)` and `BoostedRewarders(u32)` are stored in instance storage. These keys are unbounded as the numbers of LP tokens can grow indefinitely.

Impact Once the instance storage limit is reached, the contract becomes unusable for new users.

Recommendation To address this issue, it's recommended to use persistent storage for unbounded data instead of instance storage.

Developer Response The developers implemented the suggested fix only in the `veWom` storage instance. Due to the controlled nature and small amount of the assets the developers decided to maintain the `AssetPid` and `BoostedRewarders` in the instance storage.

4.1.8 V-WOM-VUL-008: Deposit functionality can be permanently broken for any new token

Severity	Medium	**Commit**	fbea044
Type	Denial of Service	**Status**	Acknowledged
File(s)	`PoolV3/core_v3.rs`, `PoolV3/pool_v3_logic.rs`, `HighCovRatioFeePoolV3/core_v3.rs`, `HighCovRatioFeePoolV3/pool_v3_logic.rs`, `DynamicPoolV3/core_v3.rs`, `DynamicPoolV3/pool_v3_logic.rs`, `CoreV3/lib.rs`		
Location(s)	quote_deposit_liquidity(), _mint_fee()		
Confirmed Fix At	N/A		

The `quote_deposit_liquidity()` function is responsible for calculating the amount of LP tokens and the liability to mint when a user deposits some assets. However, there is an issue in this function that could lead to a permanent denial of service for deposits of a particular asset. The function calculates the `lp_token_to_mint` based on the ratio of `liability_to_mint` to the current `liability`:

```
let liability = get_liability(&e, asset.clone());
if liability == 0 {
    lp_token_to_mint = liability_to_mint;
} else {
    lp_token_to_mint = (liability_to_mint * get_total_supply(&e, asset.clone())) /
    liability;
}
```

Snippet 4.5: Code snippet from the `quote_deposit_liquidity()` function.

However, it fails to account for a scenario where the `liability` is non-zero while the total supply of LP tokens is zero. This can occur due to the following sequence of events:

1. A new token is added to the pool.
2. An attacker provides liquidity for that token.
3. The attacker withdraws all their liquidity, leaving behind some fees due to the withdrawal haircut fee.
4. The attacker calls `mint_fee()` for that token.

This sequence would result in a state where `liability != 0` but `total_supply == 0`, causing `lp_token_to_mint` to be computed as zero breaking the deposits for that token.

Impact This issue can lead to a permanent denial of service for deposits of a specific asset. Once triggered, it would prevent any user from depositing that particular asset into the pool.

Recommendation The `quote_deposit_liquidity()` function should take into consideration this scenario or the `mint_fee()` function should not be allowed to increase the liability of the asset if there is not LP tokens in existence.

Developer Response The developers acknowledged the issue but will not introduce changes to the code in order to maintain consistency with the Solidity implementation.

4.1.9 V-WOM-VUL-009: Changes to reward_info are not saved in the data array

Severity	Medium	**Commit**	fbea044
Type	Logic Error	**Status**	Fixed
File(s)	`boosted_multi_rewarder_logic.rs`		
Location(s)	set_reward_rate()		
Confirmed Fix At	`https://github.com/wombat-exchange/wombat-stellar-contract/commit/8012c55bd9c06c74b053d164f54992b4815f7460`, 8012c55		

The `set_reward_rate()` function in the `boosted_multi_rewarder_logic.rs` file is responsible for updating the reward rate for a specific reward token. This function retrieves the `reward_info::RewardInfoStruct` from the persistent storage, modifies its `token_per_sec` and its `last_reward_timestamp`, but fails to save these values back into the storage.

Impact The new rate and the new last update timestamp will not be saved in the storage, causing inconsistencies in the reward distribution logic.

Recommendation Ensure that the modified `RewardInfoStruct` is saved back into the data array after making all the necessary updates.

Developer Response The developers implemented the suggested fix.

4.1.10 V-WOM-VUL-010: Incorrect Denominator in wdiv()

Severity	Medium	**Commit**	fbea044
Type	Logic Error	**Status**	Fixed
File(s)	VeWom/ds_math.rs		
Location(s)	wdiv()		
Confirmed Fix At	https://github.com/wombat-exchange/wombat-stellar-contract/commit/69666784c28453899dfade95226886d4f3063d58, 6966678		

The `wdiv()` function in the `DSMath` trait from the `veWom` contract is designed to perform a division operation with a `WAD` as the scaling factor. The issue lies in the denominator of the division operation. Instead of dividing by the input parameter `v`, the function divides by `Self::WAD`, which is incorrect. The `wdiv()` function is defined as follows:

```
fn wdiv(&self, v: Self) -> Self {
   (*self * Self::WAD + v / 2.into()) / Self::WAD
}
```

Snippet 4.6: Function `wdiv` from the `ds_math.rs` file of the `VeWom` directory.

Impact The incorrect denominator in the `wdiv()` function will lead to calculation errors. This can affect any application relying on this function for accurate division operations.

Recommendation Divide by `v` instead of `Self::WAD`.

Developer Response The developers implemented the suggested fix.

4.1.11 V-WOM-VUL-011: Incorrect implementation of swap_to_credit_quote

Severity	Medium	**Commit**	fbea044
Type	Logic Error	**Status**	Fixed
File(s)	`PoolV3/core_v3.rs`, `HighCovRatioFeePoolV3/core_v3.rs`, `DynamicPoolV3/core_v3.rs`, `CoreV3/lib.rs`		
Location(s)	swap_to_credit_quote()		
Confirmed Fix At	`https://github.com/wombat-exchange/wombat-stellar-contract/commit/1ce352a73393b08d3c827e6db3d6bf5eb07bd91e`, 3d24c77		

The `swap_to_credit_quote()` function is used to quote a swap from one token to credit. In this way, the function takes as parameters, the token assets `ax`, the liability `lx`, the delta `dx` and the amplification factor `a`. This function is designed to always return a positive number, regardless if the delta is positive or negative (tokens in or out).

However, the current implementation fails to achieve this due to a logic error in the returned value. The function takes the absolute value from the `WAD_I + a` denominator, instead of the whole polynomial:

```
pub fn swap_to_credit_quote(ax: i128, lx: i128, dx: i128, a: i128) -> u128 {
    let rx = i_wdiv(ax, lx);
    let rx_ = i_wdiv(ax + dx, lx);
    let x = rx_ - i_wdiv(a, rx_);
    let y = rx - i_wdiv(a, rx);
    // adjsut credit by 1 / (1 + A)
    ((lx * (x - y)) as u128) / i_abs(WAD_I + a)
}
```

Snippet 4.7: Function `swap_to_credit_quote()` of the `core_v3.rs` file.

Impact The function will return incorrect values when the delta of the swap `dx` is negative.

Recommendation It is advisable to get rid of the `as u128` typecast and to apply the absolute value function to all the polynomial and not only to the `WAD_I + a` denominator.

Developer Response The developers implemented the suggested fix in all the identified instances.

4.1.12 V-WOM-VUL-012: Temporary storage usage for critical data in multi-step procedures

Severity	Medium	**Commit**	fbea044
Type	Denial of Service	**Status**	Acknowledged
File(s)	`HighCovRatioFeePoolV3/pool_v3_logic.rs`, `DynamicPoolV3/pool_v3_logic.rs`		
Location(s)	withdraw(), withdraw_from_other_asset(), swap()		
Confirmed Fix At	N/A		

The `withdraw()`, `withdraw_from_other_asset()` and `swap()` operations in the `pool_v3_logic.rs` file of the `HighCovRatioFeePool` and `DynamicPool` require multiple steps in order for the operations to get completed. These functions save the necessary data, for continuing with the next step, in temporary storage. This approach is based on the assumption that users will complete all steps continuously. However, according to Soroban documentation, temporary storage should only be used for non-critical information that can be easily reproduced if deleted.

In these functions, temporary storage is used to store state information between steps, which is crucial for the correct execution of the multi-step process. This reliance on temporary storage poses a risk because if the data expires, the process cannot be completed, leading to loss of funds for users.

Reference:

`https://developers.stellar.org/docs/build/guides/storage/choosing-the-right-storage`

Impact The use of temporary storage for critical data in multi-step procedures will lead to loss of funds if the user does not complete the steps in a timely manner.

Recommendation To mitigate this issue, it is recommended to use persistent storage for critical data that is required across multiple steps in a procedure, this data can be removed from storage once user completes all the required steps.

Developer Response The developers acknowledged the issue and will not introduce changes to the code. The users are expected to perform the subsequent actions continuously.

4.1.13 V-WOM-VUL-013: Loss of unpaid rewards

Severity	Medium	**Commit**	fbea044
Type	Logic Error	**Status**	Fixed
File(s)	`multi_rewarder_per_sec_logic.rs`		
Location(s)	_on_reward()		
Confirmed Fix At	`https://github.com/wombat-exchange/wombat-stellar-contract/commit/9b6ea9c7b16b3380a550d39114e20a868ccab9b3`, 9b6ea9c		

In the `_on_reward()` function, the logic for calculating and distributing rewards to users does not account for scenarios where a user has a `reward_debt` of zero but still has `unpaid_rewards` greater than zero. This situation can arise when a user withdraws all their LP tokens, and their rewards are stored in `unpaid_rewards` due to insufficient balance at the time of withdrawal. If the user later deposits again, the current logic will result in the loss of these stored rewards. The following steps work as an example:

1. Alice deposits 100 LP tokens and her `reward_debt` is calculated and set to a non-zero value.
2. Rewards accumulate over time and Alice withdraws all her LP tokens. Due to insufficient balance in the contract, some rewards are stored in `unpaid_rewards` and her `reward_debt` is reset to zero.
3. Alice later decides to re-deposit her LP tokens. However, given that her `reward_debt` is zero the logic skips the reward distribution branch and sets her `unpaid_rewards` to zero.

Impact The impact of this issue is that users may lose rewards that they are entitled to if they withdraw all their LP tokens and later re-deposit.

Recommendation To address this issue, the logic in the `on_reward()` function should be updated to consider the `unpaid_rewards` field when determining a user's eligibility for rewards.

Developer Response The developers implemented the suggested fix.

4.1.14 V-WOM-VUL-014: Incorrect haircut calculation due to scale factor misalignment

Severity	Medium	**Commit**	fbea044
Type	Logic Error	**Status**	Acknowledged
File(s)	`DynamicPoolV3/core_v3.rs`		
Location(s)	quote_swap()		
Confirmed Fix At	N/A		

The `quote_swap()` function is designed to calculate the amount a user would receive when swapping one asset for another. In this function, a `scale_factor` is applied to adjust the `from_amount` into the units of the `to_token`, which is necessary when the tokens involved in the swap have different exchange rates.

The issue arises in the calculation of the `haircut`, which is a fee applied to the swap. The function uses the variable `temp_from_amount`, which has been scaled by the `scale_factor`, to compute the haircut. This is problematic because the haircut should be calculated based on the original `from_amount` in the units of the `from_token`, which is the one going out when `from_amount < 0`, not the scaled amount. For instance, in a swap from BNB to BNBx, where the price is 1.5 BNB per BNBx, swapping out -10 BNBx requires an input of 15 BNB. The `temp_from_amount` would be computed to -15 BNB, and the haircut would incorrectly be computed using this scaled quantity instead of the original -10 BNBx.

```
if temp_from_amount > 0 {
        // normal quote
        haircut = u_wmul(ideal_to_amount, haircut_rate);
        actual_to_amount = ideal_to_amount - haircut;
} else {
        actual_to_amount = ideal_to_amount;
        haircut = u_wmul(-temp_from_amount as u128, haircut_rate);
}
```

Snippet 4.8: Code snippet from the `quote_swap()` function. The snippet shows the computation of the `haircute` fee.

Impact The impact of this issue is that the `haircut` fee will be inaccurately calculated, leading to incorrect amounts being deducted from the user's swap.

Recommendation To address this issue, the `haircut` calculation should be performed by scaling `temp_from_amount` back to the `from_token` units.

Developer Response The developers acknowledged the issue but will not introduce changes to the code in order to maintain consistency with the Solidity implementation.

4.1.15 V-WOM-VUL-015: Centralization Risk

Severity	Low	**Commit**	fbea044
Type	Authorization	**Status**	Acknowledged
File(s)	See description		
Location(s)	See description		
Confirmed Fix At	N/A		

All the contracts in the Wombat-Exchange declare an owner account who is given special permissions. Most significantly, this owner can:

- Upgrade the implementation of contracts like: `BoostedMasterWombat`, `MultiRewarderPerSec`, `BoostedMultiRewarder` and `VeWom`.
- Set important parameters such as amplification factor, haircut rate, withdrawal fees, and add or remove support for assets, which directly affect the `Pool` contracts' operations and participant returns.
- Set an incorrect price in the `GovernedPriceFeed` contract, which will impact the behavior of the `DynamicPool` contract operations.

Impact If the private key of the owner were stolen, a hacker would have access to sensitive functionality that could compromise the project. For example, a malicious owner could drain the funds from all the contracts in the Wombat ecosystem by upgrading implementations of the contracts and by setting exploitable parameters in the pool contracts.

Recommendation As these are all particularly sensitive operations, we would encourage the developers to utilize a decentralized governance or multi-sig contract as opposed to a single account, which introduces a single point of failure.

Developer Response The developers plan to use a Safe multisig.

4.1.16 V-WOM-VUL-016: Potential Underflow in Cash and Liability Removal Functions

Severity	Low	**Commit**	fbea044
Type	Data Validation	**Status**	Fixed
File(s)	`Asset/asset_setter.rs`, `DynamicAsset/asset_setter.rs`, `PriceFeedAsset/asset_setter.rs`		
Location(s)	remove_cash_internal(), remove_liability_internal()		
Confirmed Fix At	`https://github.com/wombat-exchange/wombat-stellar-contract/commit/036e3d5af1334fc238e6144a4ed107d91e27be49`, 036e3d5		

The `remove_cash_internal()` and `remove_liability_internal()` functions in the `asset_setter.rs` file lack proper checks to ensure that the resulting balances remain non-negative after removal operations. While the current implementation checks that the input `amount` is non-negative, it does not verify that subtracting this amount from the existing balance will result in a non-negative value.

Impact While the auditors could not find a direct way to exploit this issue, its impact will lead to violating the invariant that cash and liability of an asset are non negative values.

Recommended Mitigation Steps To mitigate this issue, validate that the resulting balance after subtraction is non-negative in both `remove_cash_internal()` and `remove_liability_internal()`. This change ensures that the operation will panic if it would result in a negative balance, maintaining the invariant of non-negative cash and liability balances throughout the contract's execution.

Developer Response The developers implemented the suggested fix.

4.1.17 V-WOM-VUL-017: Potential underflow in tip_bucket_balance() can break asset cash accounting

Severity	Low	**Commit**	fbea044
Type	Data Validation	**Status**	Fixed
File(s)	`PoolV3/pool_v3_logic.rs`, `HighCovRatioFeePoolV3/pool_v3_logic.rs`, `DynamicPoolV3/pool_v3_logic.rs`		
Location(s)	tip_bucket_balance()		
Confirmed Fix At	`https://github.com/wombat-exchange/wombat-stellar-contract/commit/49d6ad4ff6d7734789c2f49d0d5c1e4c14aafb85`, 49d6ad4		

The `tip_bucket_balance()` function is responsible for calculating the balance of the tip bucket for a given token. This function plays a role in the pool's accounting system, particularly in the `fill_pool()` function which moves funds from the tip bucket to the pool to maintain the desired coverage ratio. The calculation in `tip_bucket_balance()` involves subtracting the `cash` and `fee_collected` from the underlying token balance:

```
(i_to_wad(
    client.underlying_token_balance(),
    client.underlying_token_decimals(),
) - client.cash()
    - (fee_collected as i128)) as u128
```

Snippet 4.9: Code snippet for the tip bucket balance computation from the `pool_v3_logic.rs` file.

However, this calculation does not check if the result is non-negative before casting it to `u128`.

Impact While the auditors could not find a direct way to exploit this finding, this oversight can lead to serious issues:

1. If the subtraction results in a negative value, casting it to `u128` will cause an underflow, resulting in an unexpectedly large positive value.
2. This incorrect balance will be used in the `fill_pool()` function.
3. As a result, the cash accounting for the asset will be severely disrupted.

Recommendation To address this issue, implement a check to ensure the result is non-negative before casting it to `u128`.

Developer Response The developers implemented the suggested fix.

4.1.18 V-WOM-VUL-018: Integer Square Root Functions Return Incorrect Results for Negative Inputs

Severity	Low	**Commit**	fbea044
Type	Data Validation	**Status**	Acknowledged
File(s)	`PoolV3/signed_safe_math.rs`, `HighCovRatioFeePoolV3/signed_safe_math.rs`, `DynamicPoolV3/signed_safe_math.rs`, `SignedSafeMath/lib.rs`		
Location(s)	i_sqrt(), i_sqrt_2args()		
Confirmed Fix At	N/A		

The `i_sqrt()` and `i_sqrt_2args()` functions are designed to compute the integer square root of a signed integers. However, these functions have a flaw in their implementation. Instead of panicking or handling negative inputs appropriately, they return 1 when the input value is negative. The `i_sqrt()` function is implemented as follows:

```
pub fn i_sqrt(y: i128) -> i128 {
    let mut z = 0;
    if y > 3 {
        // ... calculation for positive values ...
    } else if y != 0 {
        z = 1;
    }
    z
}
```

Snippet 4.10: Square root function implementation from the `signed_safe_math.rs` file.

In both functions, when the input value is negative, the `else if y != 0` condition is met, setting `z = 1`. This results in the function returning 1 for any negative input, which is mathematically incorrect and potentially dangerous for calculations relying on these functions.

Impact Although the auditors couldn't find a direct way to exploit this issue, the incorrect handling of negative inputs in these square root functions could lead to serious miscalculations. Future updates to the protocol might create conditions where negative numbers are provided to these functions, potentially causing significant errors.

Recommendation To address this issue, the functions should be modified to handle negative inputs correctly.

Developer Response The developers acknowledged the issue but will not introduce changes to the code in order to maintain consistency with the Solidity implementation.

4.1.19 V-WOM-VUL-019: Exchange rate might be stale in pool operations

Severity	Low	**Commit**	fbea044
Type	Logic Error	**Status**	Partially Fixed
File(s)	`PoolV3/pool_v3_logic.rs`, `HighCovRatioFeePoolV3/pool_v3_logic.rs`, `DynamicPoolV3/pool_v3_logic.rs`		
Location(s)	withdraw(), withdraw_from_other_asset(), swap(), deposit()		
Confirmed Fix At	`https://github.com/wombat-exchange/wombat-stellar-contract/commit/b6c7bc3079c241de4c64524b0f85601fda0a5e1b`, b6c7bc3		

The asset contracts maintain an exchange rate between minted LP tokens and liability. This exchange rate is used during deposits and withdrawal operations as well as the liability of a current asset is necessary to compute swap operations. This exchange rate increases whenever the collected liability (via the collected fees) is added back to the asset when `mint_fee()` function is called.

However, `mint_fee()` is not called in the deposits, withdrawal and swap workflows. Therefore, it is possible that there might be fees collected but not added as liability back to pool.

Impact As a result, an incorrect amount of liability will be used during these workflows impacting the swaps computations as well as the LP tokens minted or burned during deposits/withdrawals.

This issue impacts both `PoolV3` and `HighCovRatioFeePoolV3`.

Recommendation Mint the collected fees previous any pool operation or document the requirement to call `mint_fee()` manually constantly.

Developer Response The developers decided to fix the issue in the `deposit`, `withdraw`, and `withdraw_from_other_asset` operations but not in the `swap` operation, to maintain equivalence with the Solidity implementation.

4.1.20 V-WOM-VUL-020: Incorrect reward calculation in notifyRewardAmount leads to unfair reward distribution

Severity	Low	**Commit**	fbea044
Type	Logic Error	**Status**	Acknowledged
File(s)	`boosted_master_wombat_logic.rs`		
Location(s)	notify_reward_amount()		
Confirmed Fix At	N/A		

The `notify_reward_amount()` function in the `BoostedMasterWombat` contract is responsible for distributing `wom` tokens as rewards over a 7-day period. However, the current implementation contains a flaw that can lead to unfair reward distribution and potential dilution of rewards for existing stakers.

The function is designed to calculate the new reward rate based on the provided amount and any leftover rewards from the previous period. However, it fails to update the pool's state before performing these calculations. This oversight results in already distributed rewards being incorrectly counted as leftovers and included in the next reward period. The root cause of this issue lies in the absence of a call to `_update_pool()`. Without updating the pool's state, the function use an outdated value for `last_reward_timestamp`, leading to an incorrect calculation of leftover rewards.

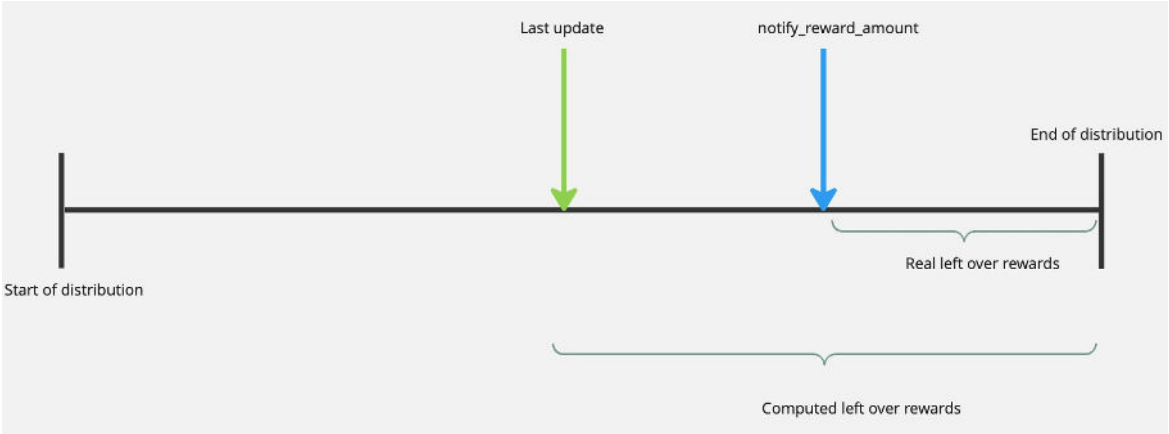

Figure 4.1: Visual description of the computation of left over rewards during `notify_reward_amount`.

Impact This issue has an impact on the fairness and accuracy of the reward distribution system as existing stakers will need to wait more time to get their already entitled rewards and they may receive fewer rewards as new stakers can come in after `notify_reward_amount()`.

Recommendation Update pool before computing the left over rewards.

Developer Response The developers acknowledged the issue but will not introduce changes to the code in order to maintain consistency with the Solidity implementation.

4.1.21 V-WOM-VUL-021: Unsafe typecasts in the codebase

Severity	Low	**Commit**	fbea044
Type	Overflow	**Status**	Acknowledged
File(s)	See description		
Location(s)	See description		
Confirmed Fix At	N/A		

In several parts of the codebase the logic relies on unsafe typecasts of the form: `as i128/u128` without any type of validation of the value to be converted.

Impact While the auditors could not find a direct way to exploit these unsafe typecasts, their impact will lead to unexpected behavior, including the violation of several invariants of the codebase.

Recommendation Implement safe typecasting by adding validation checks before performing the cast. Ensure that the value being cast is within the acceptable range of the target type.

It is important to note that the Solidity contracts implement safe typecasting functions, see reference here.

Developer Response The developers acknowledge the issue but will not introduce code changes.

4.1.22 V-WOM-VUL-022: Multi rewarder upgrade should only happen with owner approval

Severity	Low	**Commit**	fbea044
Type	Access Control	**Status**	Fixed
File(s)	`boosted_multi_rewarder_logic.rs`		
Location(s)	upgrade()		
Confirmed Fix At	`https://github.com/wombat-exchange/wombat-stellar-contract/commit/828ffd3e44f8c5ea2595460e7503d1cc980e1dcc`, 828ffd3		

The `upgrade()` function of the `boosted_multi_rewarder` contract is protected by the operator role:

```
pub fn upgrade(e: &Env, admin: Address, new_wasm_hash: BytesN<32>) {
    admin.require_auth();
    has_operator_role(e, admin);
    ....
}
```

Snippet 4.11: Code snippet from the `upgrade()` function in `boosted_multi_rewarder_logic.rs`.

Although this access control is flawed as described in V-WOM-VUL-001, this function should be protected to only be callable by the owner of the contract.

Impact Anyone with the operator role will be able to upgrade `boosted_multi_rewarder` contract allowing these account sto drain all the funds from the contract.

Recommendation The `upgrade()` function should only be allowed to be called by the owner of the contract.

Developer Response The developers implemented the suggested fix.

4.1.23 V-WOM-VUL-023: Data inconsistency due to outdated data array

Severity	Low	**Commit**	fbea044
Type	Logic Error	**Status**	Fixed
File(s)	`boosted_multi_rewarder_logic.rs`		
Location(s)	_add_reward_token(), set_reward_rate()		
Confirmed Fix At	`https://github.com/wombat-exchange/wombat-stellar-contract/commit/779f9b58f26938ff1a4f92e277d1413c6c29b5b4`, 779f9b5		

The `set_reward_rate()` function in the `boosted_multi_rewarder_logic.rs` file is responsible for updating the reward rate for a specific token. This function interacts with the vector of `RewardInfoStruct`'s stored in persistent storage.

The function begins by retrieving a copy of this vector from persistent storage into a local variable `data_array`. It then calls the `_update_reward()` function, which updates the reward information in persistent storage. However, the local `data_array` does not reflect these updates because it was retrieved before `_update_reward()` was called.

Consequently, when `data_array` is written back to persistent storage at the end of `set_reward_rate()`, it overwrites any changes made by `_update_reward()`.

Impact The impact of this issue is that any updates made to the reward information by `_update_reward()` are lost when `set_reward_rate()` writes the outdated `data_array` back to persistent storage. Therefore, this will lead to the loss of all the rewards accrued from the `last_reward_timestap`.

It is important to consider that a similar scenario happens in the `_add_reward_token()` function.

Recommendation To resolve this issue, ensure that `_update_reward()` is called at the very beginning of `set_reward_rate()` and `_add_reward_token()`, before retrieving the `data_array` from persistent storage.

Developer Response The developers implemented the suggested fix.

4.1.24 V-WOM-VUL-024: Unauthorized access to return functions can disrupt intended user operations

Severity	Low	**Commit**	fbea044
Type	Authorization	**Status**	Fixed
File(s)	`HighCovRatioFeePool/pool_v3_logic.rs`, `DynamicPool/pool_v3_logic.rs`		
Location(s)	return_withdraw_from_other_asset_p1(), return_swap_p1()		
Confirmed Fix At	`https://github.com/wombat-exchange/wombat-stellar-contract/commit/461d72ce7cd4040b7abf14b9b62009e7e2111de8`, 461d72c		

The `withdraw_from_other_asset()` function allows users to initiate a withdrawal process. This process is divided into multiple steps, and users have the option to either continue with the operation or call `return_withdraw_from_other_asset_p1()` to revert the withdrawal. However, the `return_withdraw_from_other_asset_p1()` function lacks proper authorization checks, specifically `user.require_auth()`. This means that any malicious actor could potentially call this function and disrupt the withdrawal process for another user. The same issue is present in the `return_swap_p1()` function, which is part of the swap operation process. These functions should be protected to ensure that only the user who initiated the operation can revert it.

Impact The lack of authorization checks in these function can lead to unauthorized access, allowing malicious actors to disrupt the intended operations of legitimate users.

Recommendation To mitigate this issue, implement proper authorization checks in the `return_withdraw_from_other_asset_p1()` and `return_swap_p1()` functions. Ensure that these functions can only be called by the user who initiated the respective operation.

Developer Response The developers implemented the suggested fix.

4.1.25 V-WOM-VUL-025: Potential loss due to non-atomic withdrawal and swap operations in withdraw from other asset

Severity	Low	**Commit**	fbea044
Type	Usability Issue	**Status**	Acknowledged
File(s)	`HighCovRatioFeePool/pool_v3_logic.rs`, `DynamicPool/pool_v3_logic.rs`		
Location(s)	withdraw_from_other_asset()		
Confirmed Fix At	N/A		

The `withdraw_from_other_asset()` function in the `pool_v3_logic.rs` file of the `HighCovRatioFeePool` is designed to allow users to withdraw a certain amount of one asset and swap it for another. However, the withdrawal and swap operations are not executed atomically, instead they are executed in multiple steps.

The function currently relies solely on the `minimum_amount` parameter for the swap operation as slippage protection for the user, without considering a minimum amount for the withdrawal itself. This means that if a user decides not to proceed with the swap due to unfavorable conditions such as slippage (in the swap), they are not protected by a minimum withdrawal amount. As a result, users may end up with less than expected if the swap does not occur as intended.

```
1 (from_amount_in_wad, temp) = _withdraw(e, from_asset, liquidity, 0);
```

Snippet 4.12: Code snippet from the `withdraw_from_other_asset()` function. It shows that the `0` argument is passed as the minimum amount.

Impact The lack of a minimum withdrawal amount can lead to a situation where users receive less than anticipated if they choose not to proceed with the swap or if the swap fails due to slippage.

Recommendation To mitigate this issue, it is recommended to introduce a separate minimum amount parameter for the withdrawal operation. This would ensure that users are protected by a minimum withdrawal amount, regardless of whether the swap is executed.

Developer Response The developers acknowledge the issue but will not introduce code changes.

4.1.26 V-WOM-VUL-026: Inconsistent Tip Bucket Balance Due to Multi-step Procedures

Severity	Low	Commit	fbea044
Type	Logic Error	Status	Acknowledged
File(s)	`HighCovRatioFeePool/pool_v3_logic.rs`, `DynamicPool/pool_v3_logic.rs`		
Location(s)	tip_bucket_balance()		
Confirmed Fix At	N/A		

The `withdraw()`, `withdraw_from_other_asset()`, and `swap()` functions in the `pool_v3_logic.rs` file of the high coverage fee pool involve multi-step procedures that can lead to inconsistencies in the tip bucket balance. These functions interact with the asset contracts by adjusting the cash, liability and token balances depending on the type of operation. However, the operations are not atomic, meaning that the cash balance might be adjusted in one step while the token balance is adjusted in another. This non-atomicity can result in a temporary inflation of the tip bucket balance, as the balance relies on both the token and cash balances being accurate at all times.

For example, during a multi-step withdrawal, the cash balance is reduced in the first step, but the token balance is only adjusted in a subsequent step.

Impact The discrepancy will lead to an inflated tip bucket balance, which could be exploited by trusted actors through the `fill_pool()` and `transfer_tip_bucket()` functions.

Recommendation Although the current setup limits this exploitation to trusted actors, it is advisable to either fix the issue by tracking the funds pending to be transferred as the result of a withdraw or swap operation in order to not account for them in the tip bucket balance or to document this behavior to prevent potential issues in the future.

Developer Response The developers acknowledged the issue and will not introduce changes to the code. The users are expected to perform the subsequent actions continuously.

4.1.27 V-WOM-VUL-027: During a swap, the toAsset paused is not checked

Severity	Low	**Commit**	fbea044
Type	Logic Error	**Status**	Fixed
File(s)	PoolV3/pool_v3_logic.rs, HighCovRatioFeePoolV3/pool_v3_logic.rs, DynamicPoolV3/pool_v3_logic.rs		
Location(s)	swap()		
Confirmed Fix At	https://github.com/wombat-exchange/wombat-stellar-contract/commit/9a978e4048ecdb5542768c9d6c24cd91effb0621, 9a978e4		

The pools perform swaps in the `swap()` function. This function checks if the `from` asset is not paused. This function does not validate if the `to` asset is not paused

```
pub fn swap(
    e: &Env,
    user: Address,
    from_token: Address,
    to_token: Address,
    from_amount: u128,
    minimum_to_amount: u128,
    to: Address,
    deadline: u128,
) -> (u128, u128) {
    e.storage()
        .instance()
        .extend_ttl(INSTANCE_LIFETIME_THRESHOLD, INSTANCE_BUMP_AMOUNT);
    when_not_paused(e);
    user.require_auth();

    let mut actual_to_amount;
    let mut haircut;
    _check_same_address(from_token.clone(), to_token.clone());
    if from_amount == 0 {
        panic!("WOMBAT_ZERO_AMOUNT()");
    }
    _ensure(e, deadline);
    required_asset_not_paused(e, from_token.clone());
```

Snippet 4.13: Snippet from `swap()`

Impact Cash may be swapped out when to asset is paused

Recommendation Add a check to swap function that the `to` asset is not paused.

Developer Response The developers implemented the suggested fix.

4.1.28 V-WOM-VUL-028: fill_pool may put the protocol out of equilibrium

Severity	Low	Commit	fbea044
Type	Logic Error	Status	Acknowledged
File(s)	PoolV3/pool_v3_logic.rs, HighCovRatioFeePoolV3/pool_v3_logic.rs, DynamicPoolV3/pool_v3_logic.rs		
Location(s)	fill_pool()		
Confirmed Fix At	N/A		

The protocol needs to maintain the global ratio `r*` to 1. The function `fill_pool()` is used to add the cash back to the pool to compensate for the errors accumulated during integer calculations and square roots.

```
pub fn fill_pool(e: &Env, token: Address, amount: u128) {
    e.storage()
        .instance()
        .extend_ttl(INSTANCE_LIFETIME_THRESHOLD, INSTANCE_BUMP_AMOUNT);

    _only_dev(e);
    let asset = _asset_of(e, token.clone());
    let client_asset = asset::Client::new(&e, &asset);

    let tip_bucket_bal = tip_bucket_balance(e, token.clone());

    if amount > tip_bucket_bal {
        // revert if there's not enough amount in the tip bucket
        panic!("WOMBAT_INVALID_VALUE()");
    }

    client_asset.add_cash(&(amount as i128));
    e.events()
        .publish((token, amount), symbol_short!("FillPool"));
}
```

Snippet 4.14: Snippet from `fill_pool()`

However, the function fill pool accepts the amount as an argument and add that amount of cash to the pool. The amount arg should have an upper bound that is the amount required to bring the pool back into equilibrium.

Impact This function can be called with more amount than required which will put the system out of equilibrium.

Recommendation Add a validation that the `amount` variable has an upper bound that is the required cash to bring the pool back to equilibrium.

Developer Response The developers acknowledged the issue but will not introduce changes to the code in order to maintain consistency with the Solidity implementation.

4.1.29 V-WOM-VUL-029: Stale price data in GovernedPriceFeed

Severity	Low	Commit	fbea044
Type	Data Validation	Status	Acknowledged
File(s)	`GovernedPriceFeed/logic.rs`		
Location(s)	get_latest_price()		
Confirmed Fix At	N/A		

The `get_latest_price()` function is responsible for retrieving the latest price of a token from the storage. The function checks if the provided token matches the stored token before returning the price. However, the function does not verify the freshness of the price data by checking the `LastUpdate` timestamp. This means that the price returned could be outdated or stale, as there is no mechanism to ensure that the price has been updated recently.

Additionally, the function does not return the `LastUpdate` timestamp, which would allow the caller to independently verify the freshness of the price data.

Impact The lack of verification for the `LastUpdate` timestamp in the `get_latest_price()` function can lead to the use of stale price data.

Recommendation To address this issue, it is recommended to implement a check for the `LastUpdate` timestamp within the `get_latest_price()` function. This check should ensure that the price data is recent enough to be considered valid. Additionally, the function should return the `LastUpdate` timestamp along with the price, allowing the caller to assess the freshness of the data.

Developer Response The developers acknowledged the issue but will not introduce changes to the code in order to maintain consistency with the Solidity implementation.

4.1.30 V-WOM-VUL-030: Incorrect cash check due to scale factor misalignment

Severity	Low	**Commit**	fbea044
Type	Data Validation	**Status**	Acknowledged
File(s)	DynamicPoolV3/core_v3.rs		
Location(s)	quote_swap()		
Confirmed Fix At	N/A		

The `quote_swap()` function is responsible for calculating the amount a user would receive when swapping one asset for another. In the `DynamicPool`, the logic needs to account for the fact that the tokens involved in the swap may not be pegged 1:1. For example, `xBNB` and `stkBNB` may have different prices in terms of `BNB`. To address this, a `scale_factor` in units of `to_token / from_token` is introduced to transform `from_amount` into units of `to_token`:

```
if scale_factor != WAD {
   // apply scale factor on from-amounts
   from_cash = (from_cash * (scale_factor as i128)) / WAD_I;
   from_liability = (from_liability * (scale_factor as i128)) / WAD_I;
   temp_from_amount = (temp_from_amount * (scale_factor as i128)) / WAD_I;
}
```

Snippet 4.15: Code snippet from the `quote_swap()` function. The code computes `from` values in terms of the `to` token.

At the end of the function's logic, the code validates that there is actually enough cash of the token going out:

```
if (temp_from_amount > 0 && to_cash < (ideal_to_amount as i128))
   || (temp_from_amount < 0 &&
   get_cash(&e, from_asset.clone()) < -temp_from_amount)
{
  panic!("CORE_CASH_NOT_ENOUGH()");
}
```

Snippet 4.16: Code snippet from the `quote_swap` function. The code validates there is enough cash of the token going out.

However, an issue arises in the logic of the second clause of the `||` operator. This clause checks whether there is enough cash available for an exact-output swap. The function uses the variable `temp_from_amount`, which has been scaled by the `scale_factor`, to perform this check. However, the `get_cash()` function retrieves the cash value in the original units of the `from_asset`, not the scaled units.

Impact The impact of this issue is that the function may incorrectly determine whether there is enough cash available for a swap. When combined with the issue: V-WOM-VUL-016, it can be used potentially to drain the pool contracts.

Recommendation To resolve this issue, the cash check should be performed using the original `from_amount` instead of the scaled `temp_from_amount`. This ensures that both values are in the same units, allowing for a correct comparison.

Developer Response The developers acknowledged the issue but will not introduce changes to the code in order to maintain consistency with the Solidity implementation.

4.1.31 V-WOM-VUL-031: No way to change or renounce ownership in veWom and boosted rewarder contracts

Severity	Warning	**Commit**	fbea044
Type	Maintainability	**Status**	Fixed
File(s)	`vewom_contract.rs`, `boosted_master_wombat_contract.rs`, `boosted_multi_rewarder_contract.rs`		
Location(s)	See description		
Confirmed Fix At	`https://github.com/wombat-exchange/wombat-stellar-contract/commit/9409ac1e7f7cc4deae8c53dc6f8c18cce0f4721d`, 9409ac1		

The `veWom`,`boosted_master_wombat` and `boosted_multi_rewarder` contracts set an owner during their initialization procedures. However, these contracts are not implementing the functionality to change the owner or renounce the ownership of the contracts.

Impact It is not possible to transfer or renounce ownership of the `veWom`, `boosted_master_wombat` and `boosted_multi_rewarder` contracts.

Recommendation Implement a `transfer_ownership()` function.

Developer Response The developers implemented the suggested fix.

4.1.32 V-WOM-VUL-032: User Can Keep Exceeding the Maximum Breeding Length

Severity	Warning	**Commit**	fbea044
Type	Logic Error	**Status**	Fixed
File(s)	`vewom_logic.rs`		
Location(s)	mint_vewom()		
Confirmed Fix At	`https://github.com/wombat-exchange/wombat-stellar-contract/commit/6fdeb69fff7450e4e1880e31d1f73e7247a4db20`, 6fdeb69		

The `mint_vewom()` function in the `vewom_logic.rs` file is responsible for minting new `veWOM` tokens based on the amount of `WOM` tokens locked by a user and the duration of the lock. The function checks if the user's current breeding length is equal to the maximum allowed breeding length before proceeding. However, this check should use a greater than or equal to (>=) comparison instead of an equality (==) comparison to prevent users from exceeding the maximum breeding slots. Take the following scenario as example:

1. The contract is initialized with a maximum breeding length of 10.
2. A user locks `WOM` tokens multiple times, reaching a breeding length of 10.
3. The contract owner reduces the `MaxBreedingLength` to 5.
4. The user attempts to lock more `WOM` tokens. The current check (==) does not prevent this action because the user's breeding length (10) is not equal to the new maximum (5).

Impact Users can keep violating the maximum breeding length limit in the case of a limit reduction.

Recommendation Replace the equality check (==) with a greater than or equal to (>=) comparison.

Developer Response The developers implemented the suggested fix.

4.1.33 V-WOM-VUL-033: Inconsistent lock days validation

Severity	Info	**Commit**	fbea044
Type	Maintainability	**Status**	Fixed
File(s)	vewom_logic.rs		
Location(s)	update()		
Confirmed Fix At	https://github.com/wombat-exchange/wombat-stellar-contract/commit/19541a4c4d3bbc226842e95d3b358589b5317765, 19541a4		

The `update()` function in the `vewom_logic.rs` file, validates the `lock_days` argument in the following way:

```
1 if lock_days < DataKeyVeWom::MinLockDays.get(e) || lock_days > DataKeyVeWom::
      MaxLockDays.get(e)
2 {
3     panic!("lock days is invalid");
4 }
```

Snippet 4.17: Code snippet from the `update()` function. It shows how the validation of the `lock_days` argument.

However, there is already an implemented function for the same purpose, `valid_lock_days()`.

Impact By not using the existing `valid_lock_days` function, the code is more prone to errors and harder to maintain. If the validation logic for lock days needs to be updated, it must be changed in multiple places, increasing the risk of bugs and inconsistencies.

Recommendation Refactor the code to use the `valid_lock_days` function for validating lock days.

Developer Response The developers implemented the suggested fix.

4.1.34 V-WOM-VUL-034: Typos, redudant code and other minor issues

Severity	Info	**Commit**	fbea044
Type	Maintainability	**Status**	Fixed
File(s)	See description		
Location(s)	See description		
Confirmed Fix At	`https://github.com/wombat-exchange/wombat-stellar-contract/commit/3802fe450da80172357d44bfb08e4abcee6e6a96`, 3802fe4		

In the following locations, the auditors identified typos, redundant code, and the use of hard-coded values:

- `GovernedPriceFeed/contract.rs`: The function names `get_lastest_price` and `logic::get_lastest_price` contain a typo in `lastest`. It should be `latest`.
- `GovernedPriceFeed/logic.rs`: The functions `add_operator_role` and `remove_operator_role` use a hard-coded value `0x46a52c.....` to represent the operator role. It is recommended to define a constant for this value instead of hardcoding it.
- `VeWom/pausable_contract.rs`: The function `_unpause` contains a redundant `set` operation.

Impact These minor errors may lead to future developer confusion or mistakes.

Developer Response The developers resolved all the minor issues.

Glossary

AMM Automated Market Maker. 1

smart contract A self-executing contract with the terms directly written into code. Hosted on a blockchain, it automatically enforces and executes the terms of an agreement between buyer and seller. Smart contracts are transparent, tamper-proof, and eliminate the need for intermediaries, making transactions more efficient and secure. 45

Solidity The standard high-level language used to develop smart contracts on the Ethereum blockchain. See `https://docs.soliditylang.org/en/v0.8.19/` to learn more. 1

Soroban Soroban is the smart contracts platform on the Stellar network. See `https://stellar.org/soroban` to learn more. 1