Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Leo Wallet

Pondo Protocol

Veridise Inc.

September 9, 2024

► **Prepared For:**

Demox Labs
https://www.demoxlabs.xyz/

► **Prepared By:**

Benjamin Mariano
Alberto Gonzalez
Mark Anthony

► **Contact Us:** contact@veridise.com

► **Version History:**

Sep. 9, 2024	V2.1
Sep. 3, 2024	V2
Aug. 29, 2024	V1
Aug. 7, 2024	Initial Draft

Contents

1 Executive Summary

From Jul. 22, 2024 to Aug. 2, 2024, Demox Labs engaged Veridise to review the security of their Pondo Protocol. Pondo is a liquid staking protocol for Aleo. The protocol works by allowing people to stake tokens in exchange for Paleo tokens, which represent the staked assets. The protocol itself handles the actual staking by maintaining a pool of potential validators, the top 5 of whom are selected weekly and used as the current targets of staked funds for the protocol. Veridise conducted the assessment over 30 person-days, with 3 engineers reviewing code over 10 days on commit `8cde12d`. The auditing strategy involved extensive manual code review performed by Veridise engineers.

Project summary. The protocol is mainly divided into the following programs:

- `pondo_core_protocol`. This is the primary and user-facing part of the protocol. It supports deposits (where Aleo credits are exchanged for Paleo) and withdrawals (the reverse process). These transactions correspond to staking and unstaking funds. Additionally, it manages the "rebalancing" process, which occurs weekly and selects the 5 best-performing validators to reinvest all staked funds.
- `pondo_delegator`. This program implements one of the built-in delegators created by the Pondo team. Delegators serve as the interface for interacting with validators and handle the actual staking and unstaking of funds with these validators.
- `pondo_oracle`. This program is responsible for determining the top 5 performing validators each week.
- `pondo_reference_delegator`. This is a template program which is intended to be instantiated by users who want to suggest a new validator to be included in the considered pool of validators.
- `pondo_token`. This is token registered in the MTSP program that users can burn in exchange of pAleo earned by the protocol via comissions.
- `pondo_staked_aleo`. Token registered in the MTSP program representing the liquid staking of credits.

Code assessment. The Pondo Protocol developers provided the source code of the program for review. The source code appears to be mostly original code written by the Pondo Protocol developers. It contains some very limited documentation in the form of in-line comments in the code. No other documentation was provided to the Veridise auditors for the audit.

At the time of the audit, the code contained no automated tests. However, by the time this report was completed, the Pondo Protocol developers had added an automated test suite covering the state transitions for the core program as well as the oracle update data logic.

Summary of issues detected. The audit uncovered 23 issues, 6 of which are assessed to be of high or critical severity by the Veridise auditors. Specifically, auditors discovered that the oracle program could have been re-initialized at any time (V-PON-VUL-003); a malicious validator

could have caused a denial of service for the entire protocol (V-PON-VUL-002) or a denial of service due to an underflow (V-PON-VUL-004); a malicious user could have drained all the pAleo tokens from the Pondo token program (V-PON-VUL-001); and in certain scenarios, bonded withdrawals could have been used to break the protocol accounting (V-PON-VUL-005). The Veridise auditors also identified 5 medium-severity issues which most of them are related to an incorrect computation of rewards in the protocol (V-PON-VUL-007, V-PON-VUL-009, V-PON-VUL-010); a double accounting of validators' boost (V-PON-VUL-011); and an incorrect distribution of credits to delegators (V-PON-VUL-008). Additionally, the Veridise audits also identified 2 low-severity, 8 warnings, and 2 informational findings.

Recommendations. After auditing the protocol, the auditors had a few suggestions to improve the Pondo Protocol security.

Refactoring. The codebase can reduce several lines of code by refactoring common logic into inline functions. Specifically, the Veridise auditors noted that the logic for computing the total credits in the system is repeated across the deposit and withdrawal functions. Consolidating this repeated logic will not only make the code more concise and readable but also easier to maintain and less prone to errors.

Denial Of Service. The paradigm of the Aleo ecosystem separates program logic into off-chain executions (to preserve privacy) and on-chain executions (to modify the public ledger). This introduces several aspects to consider, such as confirming in the on-chain execution any program state that was used in the off-chain execution. For instance, in the particular case of Pondo, there are on-chain confirmations that the amount of tokens transferred in the off-chain part actually matches the total funds from the given account. This opens up vectors for denial-of-service attacks, which occur when the program's on-chain state changes, either intentionally or naturally through user interactions. Thus, it is important for Pondo developers to assess the risks of these types of attacks and their potential impact on the protocol. For example, a malicious user could send 1 micro-credit at the beginning of each block to disrupt the expected program funds.

Additional Checks Before Deployment. The Pondo Protocol implements several complex design features, including the interactions between different state configurations of the core program and the delegator programs, an oracle system for selecting top-performing validators in the entire network, and maintaining a high level of decentralization in the protocol's operation. The interaction between these components can lead to subtle issues, as seen in V-PON-VUL-002 and V-PON-VUL-005. Due to the protocol's complexity, additional care should be taken in addressing the issues found. Validating the fixes in isolation might miss the presence of other, more advanced issues introduced by code changes or overlooked during this review. Strengthening the test suite and conducting another audit cycle will help improve the protocol's security and reliability before deployment.

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
Pondo Protocol	`8cde12d`	Leo	Aleo

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Jul. 22 - Aug. 2, 2024	Manual	3	30 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Acknowledged	Fixed
Critical-Severity Issues	3	3	3
High-Severity Issues	3	3	3
Medium-Severity Issues	5	5	5
Low-Severity Issues	2	2	1
Warning-Severity Issues	8	8	4
Informational-Severity Issues	2	2	1
TOTAL	23	23	17

Table 2.4: Category Breakdown.

Name	Number
Logic Error	8
Data Validation	5
Denial of Service	3
Maintainability	3
Access Control	2
Usability Issue	2

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of the Pondo Protocol implementation in Leo. In our audit, we sought to answer questions such as:

- Can a malicious user stop the protocol operations either permanently or temporarily?
- Can a malicious user break the accounting of Aleo credits and pAleo tokens in the protocol?
- Can a malicious user gain a financial advantage over honest users using a combination of depostis and withdrawals?
- Can a malicious user get pAleo rewards without burning the expected amount of Pondo tokens?
- Does the protocol correctly compute the staking rewards and charge the expected comission?
- Can the ordering of top validators be controlled or manipulated by an attacker?
- Is it possible for a validator to get more of the protocol allocation than intended?
- Are misbehaving validators removed from contention for allocation?
- Can attackers incorrectly block well-behaved validators?
- Does the protocol correctly integrate with the MTSP and Aleo credits programs?
- Is the protocol vulnerable to common issues in Aleo/Leo programs?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved multiple auditors performing detailed manual code review.

Scope. The scope of this audit is limited to the `pondo/` folder of the source code provided by the Pondo Protocol developers, which contains the implementation of the Pondo Protocol project in Leo.

Methodology. Veridise auditors reviewed audit reports of protocols similar to Pondo Protocol and discussed the intentions of the project with the developers. They then began a manual review of the code. During the audit, the Veridise auditors met with the Pondo Protocol developers to ask questions about the code, share issues found, and clarify developer intentions.

3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

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

In this case, we judge the likelihood of a vulnerability as follows in 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

In addition, we judge the impact of a vulnerability as follows in 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

In this section, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.). Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-PON-VUL-001	pAleo tokens can be drained from the pondo toke.	Critical	Fixed
V-PON-VUL-002	A malicious validator can permanent DoS the pon.	Critical	Fixed
V-PON-VUL-003	Oracle can be re-initialized at any time	Critical	Fixed
V-PON-VUL-004	Ternary operator will cause DoS	High	Fixed
V-PON-VUL-005	Bonded withdrawals can break the protocol accou.	High	Fixed
V-PON-VUL-006	Duplicate validators and delegators handled app. .	High	Fixed
V-PON-VUL-007	Deposits will incorrectly be counted as staking. . .	Medium	Fixed
V-PON-VUL-008	Users deposits can be send to delegators in any. . .	Medium	Fixed
V-PON-VUL-009	Instant withdrawals do not update the delegated. .	Medium	Fixed
V-PON-VUL-010	Bonded withdrawals are ignored when retrieving .	Medium	Fixed
V-PON-VUL-011	Double counting of boost	Medium	Fixed
V-PON-VUL-012	rebalance redistribute can be temporary DoSed	Low	Fixed
V-PON-VUL-013	Manipulating TVL for boosting	Low	Acknowledged
V-PON-VUL-014	Initialization should revert explicitly	Warning	Fixed
V-PON-VUL-015	Users should have slippage protection during wi. .	Warning	Acknowledged
V-PON-VUL-016	A validator will not be banned during the unbon. .	Warning	Acknowledged
V-PON-VUL-017	Once banned a validator cannot be unbanned	Warning	Fixed
V-PON-VUL-018	Accidental removal of admin	Warning	Fixed
V-PON-VUL-019	Delegator allocations can sum to more than 100%	Warning	Acknowledged
V-PON-VUL-020	Centralization Risk	Warning	Fixed
V-PON-VUL-021	Lost boosting funds	Warning	Acknowledged
V-PON-VUL-022	Unused code	Info	Fixed
V-PON-VUL-023	Predefined tokens can be registered in the MTSP. .	Info	Acknowledged

4.1 Detailed Description of Issues

4.1.1 V-PON-VUL-001: pAleo tokens can be drained from the pondo token contract

Severity	Critical	**Commit**	8cde12d
Type	Logic Error	**Status**	Fixed
File(s)	`pondo_token/main.leo`		
Location(s)	finalize_burn_public()		
Confirmed Fix At	2691ab4		

The commissions, in the form of `pAleo` tokens, taken by the core protocol are sent to the `pondo` token contract. Any holder can burn pondo tokens to receive a pro-rata share of the `pAleo` in the token program. To ensure users do not obtain more `pAleo` tokens than they should for the amount of pondo burned, the contract includes the following assertion using the final reserves ratio and the user withdrawal ratio:

```
let pondo_paleo_ratio: u128 = pondo_supply_after.supply * PRECISION /
    paleo_balance_after.balance;
let withdrawal_ratio: u128 = amount * PRECISION / paleo_amount;

// Ensure that the pondo to paleo ratio is greater than the withdrawal ratio
let valid_withdrawal: bool = pondo_paleo_ratio >= withdrawal_ratio;
assert(valid_withdrawal);
```

Snippet 4.1: Code snippet from the `finalize_burn_public` function.

The code validates that the resultant ratio is greater than or equal to the withdrawal ratio. However, the ratios being used are in `pondo / pAleo` units. Thus, pondo becomes more valuable as the ratio becomes smaller, not greater.

Impact This oversight allows any user to drain all the `pAleo` in the pondo token contract without owning all the pondo tokens. Consider the following scenario:

- `total_pondo = 100`
- `total_pAleo = 100`

The ratio is 1:1, meaning that a user should get at most 1 `pAleo` for each `pondo` burned.

- User burns 1 `pondo` for 99 `pAleo`, which theoretically should not be allowed. The ratios are computed as:
  - `pondo_paleo_ratio` = (100 - 1) / (100 - 99) = 99
  - `withdrawal_ratio` = 1 / 99
- The assertion passes.

Recommendation Change the comparison operator from >= to <= .

Developer Response Developers implemented the suggested fix.

4.1.2 V-PON-VUL-002: A malicious validator can permanent DoS the pondo protocol

Severity	Critical	**Commit**	8cde12d
Type	Denial of Service	**Status**	Fixed
File(s)	`pondo_delegator/main.leo`		
Location(s)	finalize_bond_failed()		
Confirmed Fix At	2d15b77		

One of the important properties of the delegators' programs is that all delegators should be able to return to the terminal state so the staking cycle can start again. If a delegator is unable to return to the terminal state, then the protocol will not be able to continue its operations.

One of these states is the `BOND_ALLOWED` state, which has 3 transitions:

1. `bond`: Makes the delegator bond to a validator and transition to the `unbond_not_allowed` state.
2. `insufficient_balance`: Takes the delegator back to the `terminal` state if the delegator does not reach the minimum amount of 10,000 credits.
3. `bond_failed`: Also takes the delegator back to the `terminal` state when the bond fails for any reason besides the minimum bond allowed.

The `bond_failed` transition assumes that the validator to which the delegator is supposed to bond must be in the committee. This is observed in the following snippet where the code uses `get` instead of `get_or_use`:

```
let validator_committee_state: committee_state = credits.aleo/committee.get(
    current_validator_state.validator);
```

Snippet 4.2: Code snippet from the `finalize_bond_failed()` function.

The assumption is that if the validator is not in the committee, then bonding always succeeds so the delegator can leave the `BOND_ALLOWED` state by calling the `bond` transition. However, there are two reasons that bonding can fail even if the validator is not in the committee:

1. Reaching the limit of 100k delegators.
2. The validator has an unbonding.

While the first scenario is unlikely to happen soon (as reaching 100k delegators would mean bonding 1B credits), the second scenario can be manipulated by a malicious validator. Here's how:

1. Have an active `validator1`.
2. Make the pondo protocol choose `validator1` to bond one of its delegators.
3. When the delegator is in the `BOND_ALLOWED` state, request an unbonding that makes `validator1` to fall below the minimum of 100 credits of self-bonding.

After step 3, `validator1` will not be in the committee and will have an active unbonding, preventing the delegator from calling the the `bond` and `bond_failed` transitions. Now, `validator1` can carry out the following actions to maintain an active unbonding:

1. Bond at least 20k credits to another controlled account.

2. Maintain an active unbonding by unbonding 1 credit before the claim deadline from the previous unbonding is reached.
3. Repeat step 5.

Impact A malicious validator can cause a delegator to get stuck in the `BOND_ALLOWED` state, preventing the pondo protocol from making any progress.

Recommendation Use `get_or_use` instead of `get` to handle the `bond_failed` transition. This will ensure that the validator committee state is correctly retrieved or handled appropriately even when the validator is not in the committee.

Developer Response The developers fixed the issue by allowing the `prep_rebalance` transition to go from the `BOND_ALLOWED` state back to the `TERMINAL` state.

4.1.3 V-PON-VUL-003: Oracle can be re-initialized at any time

Severity	Critical	**Commit**	8cde12d
Type	Access Control	**Status**	Fixed
File(s)	`pondo_oracle/main.leo`		
Location(s)	initialize()		
Confirmed Fix At	4537aed		

Oracle initialization does a number of things, including setting the control addresses, the delegator allocation amounts, and initializing the top validators to the one group address. Nothing prevents this function from being called multiple times, including in the midst of the protocol.

Impact Calling this function during operating of the protocol could lead to lost funds, as the validators will be overwritten. Furthermore, it could lead to reinstating control addresses which should be disabled or overwriting the intended delegator allocation amounts.

Recommendation Only allow this function to be called once.

Developer Response The developers have added a check which ensures that `initialize()` can only be called successfully once.

4.1.4 V-PON-VUL-004: Ternary operator will cause DoS

Severity	High	**Commit**	8cde12d
Type	Denial of Service	**Status**	Fixed
File(s)	`pondo_core_protocol/main.leo`		
Location(s)	deposit_public_as_signer, deposit_public, withdraw_public		
Confirmed Fix At	0148b52		

When computing the `deposit_pool` variable, the code needs to consider the contract's state. If the contract is in the `REBALANCING_STATE`, then the variable `currently_delegated` needs to be subtracted. This can be observed in the following code snippet:

```
let deposit_pool: u64 = current_state != REBALANCING_STATE
    ? core_protocol_account - credits_deposit - reserved_for_withdrawal
    : core_protocol_account - currently_delegated - credits_deposit -
   reserved_for_withdrawal;
```

Snippet 4.3: Code snippet from the `finalize_deposit_public_as_signer` function.

This adjustment is necessary because, in the `REBALANCING_STATE`, the pondo core contract actually has the `currently_delegated` funds within its balance, so these funds are subtracted to avoid double accounting. However, given the nature of Aleo programs, this branch will be executed even if the contract is not in the `REBALANCING_STATE`.

Impact If this branch is executed during a state than `REBALANCING_STATE`, then `core_protocol_account` will not have the `currently_delegated`funds, and the subtraction will lead to a revert due to underflow.

Recommendation Do not use a ternary operator to compute `deposit_pool`. Instead, use explicit `if` and `else` branches.

Developer Response The developers disallowed the deposits and withdrawals during the `REBALANCING` state, as a consequence, the ternary operation was removed fixing this issue.

4.1.5 V-PON-VUL-005: Bonded withdrawals can break the protocol accounting

Severity	High	**Commit**	8cde12d
Type	Logic Error	**Status**	Fixed
File(s)	`pondo_core_protocol/main.leo`		
Location(s)	withdraw_public()		
Confirmed Fix At	0148b52		

When the protocol is in the `REBALANCING_STATE`, the code does not account for non-zero `BONDED_WITHDRAWALS` . This leads to two primary issues:

1. During the rebalancing state, the `total_delegated` variable is zero as delegators do not have funds. Consequently, the bonded withdrawals carried from `rebalance_retrieve_credits` are ignored when computing the `currently_delegated`funds during deposits and withdrawals.
2. All the bonded withdrawals requested while the core program is in the rebalancing state will be ignored when `rebalance_redistribute` is called, as the `DELEGATED_BALANCE` variable will be updated with the value of all the transferred funds.

Refer to the proof of concept for both scenarios: Bonded withdrawals breaking protocol accounting.

Impact The accounting of `aleo` and `pAleo` in the system will be broken.

Recommendation To address the issues caused by `BONDED_WITHDRAWALS` during the rebalancing state, the following changes are advisable:

- Convert all bonded withdrawals to claimable withdrawals during `rebalance_retrieve_credits` .
- Implement a check to disallow bonded withdrawals when the contract is in `REBALANCING_STATE` .

Developer Response The developers implemented the suggested fix.

4.1.6 V-PON-VUL-006: Duplicate validators and delegators handled appropriately

Severity	High	**Commit**	8cde12d
Type	Logic Error	**Status**	Fixed
File(s)	`pondo_oracle/main.leo`		
Location(s)	finalize_update_data()		
Confirmed Fix At	7f74643		

After data associated with a delegator is updated, the oracle reorders the top 10 list of delegators by iteratively comparing two elements at a time (starting with the new element and the top performing delegator) and moving down the list, each time comparing the loser of the previous comparison with the next entry in the list. To avoid duplicates, the comparison function `swap_validator_data()` has an `auto_swap` functionality which will make any duplicate delegator or validator be removed from the list. However, because the comparison is only performed between the loser of the previous comparison and the next element in the top performing list, there is no guarantee that the correct delegators will be compared, triggering the `auto_swap` functionality. It is therefore possible to have duplicate delegators and duplicate validators in the top 10 list.

Another closely related concern which most be considered is *when* the validator data is updated with respect to the swapping. If the update happens before the swapping, the new data could unintentionally remain in the top list even if the validator is no longer one of the top performing validators.

Impact This could result in a single validator or delegator receiving more than the desired share of Pondo assets.

Recommendation Add logic that ensures no duplicate entries can be added to the top performing list.

Developer Response The developers implemented the suggested fix.

4.1.7 V-PON-VUL-007: Deposits will incorrectly be counted as staking profits

Severity	Medium	**Commit**	8cde12d
Type	Usability Issue	**Status**	Fixed
File(s)	`pondo_core_protocol/main.leo`		
Location(s)	distribute_deposits		
Confirmed Fix At	16a85c7		

The `distribute_deposits` function is used to transfer funds from the core contract to the delegators. However, this function does not update the `DELEGATED_BALANCE` variable. As a result, the difference between the credits managed by the delegators (including the previously sent funds) and the `DELEGATED_BALANCE` variable will be counted as staking profits.

Impact The funds sent to delegators via `distribute_deposits` will be counted as rewards when they were just users' deposits.

Recommendation It is advisable to update the `DELEGATED_BALANCE` variable in the `distribute_deposits` function.

Developer Response The developers implemented the suggested fix.

4.1.8 V-PON-VUL-008: Users deposits can be send to delegators in any proportion

Severity	Medium	**Commit**	8cde12d
Type	Data Validation	**Status**	Fixed
File(s)	`pondo_core_protocol/main.leo`		
Location(s)	distribute_deposits()		
Confirmed Fix At	5a4a078		

The Pondo Core program allocates funds to delegators in established proportions. However, the `distribute_deposits`function does not check the transferred funds against the configured proportions.

Impact Any user can use the `distribute_deposits` function to allocate a greater percentage of funds to a validator than the established proportion.

Recommendation Use a similar approach as in `rebalance_redistribute` to ensure the proper allocation of funds according to the configured proportions.

Developer Response The developers implemented the suggested fix.

4.1.9 V-PON-VUL-009: Instant withdrawals do not update the delegated funds

Severity	Medium	**Commit**	8cde12d
Type	Logic Error	**Status**	Fixed
File(s)	`pondo_core_protocol/main.leo`		
Location(s)	instant_withdraw_public()		
Confirmed Fix At	3cc6b66		

The core protocol computes staking profits by taking the difference between the current total credits managed by delegators and the `DELEGATED_BALANCE` variable. The difference is assigned to the `currently_delegated` variable, which should be used to update `DELEGATED_BALANCE` to avoid counting the rewards again.

However, the `instant_withdraw_public` function fails to update the `DELEGATED_BALANCE` variable. This oversight allows the commission to be charged repeatedly until `DELEGATED_BALANCE` is correctly updated by another function in the program.

Impact A malicious user can repeatedly call `instant_withdraw_public` with negligible amounts, causing the program to charge the rewards commission multiple times.

Recommendation It is advisable to update `DELEGATED_BALANCE` with the new `currently_delegated` variable.

Developer Response The developers implemented the suggested fix.

4.1.10 V-PON-VUL-010: Bonded withdrawals are ignored when retrieving credits

Severity	Medium	**Commit**	8cde12d
Type	Logic Error	**Status**	Fixed
File(s)	`pondo_core_protocol/main.leo`		
Location(s)	rebalance_retrieve_credits()		
Confirmed Fix At	0148b52		

The system computes staking profits by taking the difference between the credits managed by delegators, assigned to `full_balance`, and the last snapshot of this value, assigned to `current_balance`. These profits are calculated during operations involving an inflow or outflow of credits, such as in the `rebalance_retrieve_credits` function:

```
let rewards: i64 = full_balance > current_balance ? full_balance as i64 -
    current_balance as i64 : 0i64;
```

Snippet 4.4: Code snippet from the `finalize_rebalance_retrieve_credits function`. It computes the `rewards` using the difference between `full_balance` and `current_balance`.

However, `full_balance` is not taking into consideration the bonded withdrawals, which when requested have a reducing impact on `current_balance`.

Impact Users can manipulate the system by requesting bonded withdrawals before `rebalance_retrieve_credits` is called, creating an artificial difference between `full_balance` and `current_balance`. This causes the program to incorrectly charge a commission on the artificially inflated difference.

Recommendation Consider bonded withdrawals when computing `full_balance` in the `finalize_rebalance_retrieve_credits` function.

Developer Response The developers implemented the suggested fix.

4.1.11 V-PON-VUL-011: Double counting of boost

Severity	Medium	**Commit**	8cde12d
Type	Logic Error	**Status**	Fixed
File(s)	`pondo_oracle/main.leo`		
Location(s)	finalize_update_data()		
Confirmed Fix At	2691ab4		

Boost is intended to allow validators to "bribe" the protocol to have more Pondo assets delegated to them. This is intended to account for sources of profit outside of those directly tracked by Pondo (such as MEV). The boost amount is calculated by "normalizing" it and then adding it to the microcredits yield per epoch as shown in the following snippet.

```
let new_microcredits_yield_per_epoch: u128 = did_update_last_epoch ? yield_per_epoch
    + normalized_boost_amount : 0u128;
```

Snippet 4.5: Snippet from `finalize_update_data()`

It is additionally accounted for in `swap_validator_data()` where the boost amount is used to compute yields as shown in the snippet below.

```
let new_microcredits_yield_per_epoch: u128 = did_update_last_epoch ? yield_per_epoch
    + normalized_boost_amount : 0u128;
```

Snippet 4.6: Snippet from `swap_validator_data()`

Impact The boost is double counted in this case, which will allow boost to have an outsized impact on the computation.

Recommendation Account for boost only in `swap_validator_data()` by setting the `boost` field for the validator data to contain the normalized boost amount.

Developer Response The developers have implemented the suggested fix.

4.1.12 V-PON-VUL-012: rebalance redistribute can be temporary DoSed

Severity	Low	**Commit**	8cde12d
Type	Denial of Service	**Status**	Fixed
File(s)	pondo_core_protocol/main.leo		
Location(s)	rebalancing_redistribute		
Confirmed Fix At	d5def0f		

The `rebalance_redistribute` function in the pondo protocol aims to ensure that the liquidity pool stays within a targeted optimal liquidity range. The current implementation enforces this by checking if the `liquidity_pool` is between `optimal_liquidity` and `optimal_liquidity + 250` credits:

```
assert(liquidity_pool >= optimal_liquidity); // ensure that liquidity pool is at
    least optimal liquidity
assert(liquidity_pool <= optimal_liquidity + 250u64);
```

Snippet 4.7: Code snippet from the `rebalance_redistribute()` function.

However, this fixed range can lead to a denial of service (DoS) attack if deposits greater than 250 credits are made, causing the `rebalance_redistribute` function to fail.

Impact An attacker can intentionally deposit more than 250 credits, causing the `liquidity_pool` to exceed `optimal_liquidity + 250`, which will trigger the assert statement and revert the transaction.

Recommendation To mitigate this issue, it is advisable to replace the fixed margin with a percentage-based margin.

Developer Response The developers increased the margin to 10,000 credits making the attack to cost more.

4.1.13 V-PON-VUL-013: Manipulating TVL for boosting

Severity	Low	**Commit**	8cde12d
Type	Logic Error	**Status**	Acknowledged
File(s)	`pondo_oracle/main.leo`		
Location(s)	finalize_update_data()		
Confirmed Fix At	N/A		

When updating data for a delegator, the impact of boosting is "normalized" using the TVL of Pondo as shown below.

```
let boost_amount: u128 = boost.epoch == current_epoch ? boost.boost_amount as u128 :
    0u128;
// Normalize the boost amount by the pondo tvl
let current_pondo_tvl: u128 = pondo_tvl.get_or_use(0u8, 10_000_000_000_000_000u64) as
     u128; // use a high default, 10B credits
// The normalized boost amount is the amount of boost per 10K credits staked
// Note: This precision is 1 credit on a TVL of 10M credits
let normalized_boost_amount: u128 = boost_amount * PRECISION / current_pondo_tvl as
    u128;
```

Snippet 4.8: Snippet from `finalize_update_data()`

The normalized boost amount is computed by multiplying the boost amount by a constant and then dividing by the current TVL. Thus, the lower the TVL, the higher the normalized boost (and vice-versa).

Impact The TVL can be relatively easily manipulated by an attacker, thus changing the impact of boosting. For example, a user can boost immediately in a new epoch and then deposit a large amount to increase TVL. This will increase the impact of their boost vs. others who boost after the large deposit.

Recommendation Use some mechanism for normalizing boost that is not as susceptible to manipulation (or at least impose a large cost on any attacker who would want to use this for gain).

Developer Response Acknowledged.

4.1.14 V-PON-VUL-014: Initialization should revert explicitly

Severity	Warning	**Commit**	8cde12d
Type	Maintainability	**Status**	Fixed
File(s)	`pondo_core_protocol/main.leo`		
Location(s)	initialize()		
Confirmed Fix At	7c3bf77		

The `initialize` function does not explicitly check if the initialization has occurred twice. Instead, it relies on the `MTSP`contract to revert when attempting to register the `pAleo` token again.

Impact To the current auditors' knowledge, this issue cannot be exploited and therefore does not have any direct impact on the program. However, relying on external programs' assumptions can hurt the maintainability and reusability of the pondo core program.

Recommendation It is advisable to implement an explicit revert in case `initialize` is called more than once instead of relying on external programs.

Developer Response The developers implemented the suggested fix.

4.1.15 V-PON-VUL-015: Users should have slippage protection during withdrawals

Severity	Warning	**Commit**	8cde12d
Type	Data Validation	**Status**	Acknowledged
File(s)	`pondo_delegator/main.leo`		
Location(s)	instant_withdraw_public		
Confirmed Fix At	N/A		

The `finalize_instant_withdraw_public` function includes a check to ensure that the user does not withdraw more credits than allowed for a given amount of burned `pAleo`:

```
1 assert(withdrawal_credits <= withdrawal_calculation as u64);
```

Snippet 4.9: Code snippet from `finalize_instant_withdraw_public()` function. It avoids the protocol to giving more credits that it should.

However, this check only protects the protocol from giving out more credits than it should, and does not prevent users from mistakenly requesting significantly fewer credits than they are entitled to.

Impact Users may inadvertently receive fewer credits than the burned `pAleo` is worth if they provide an incorrect `withdrawal_credits` value that is too small.

Recommendation Implement an additional check to ensure that `withdrawal_credits` is not significantly less than `withdrawal_calculation`. This will prevent users from receiving substantially fewer credits than they are entitled to, improving user experience.

Developer Response Acknowledged

4.1.16 V-PON-VUL-016: A validator will not be banned during the unbond allowed state

Severity	Warning	**Commit**	8cde12d
Type	Data Validation	**Status**	Acknowledged
File(s)	`pondo_delegator/main.leo`		
Location(s)	finalize_terminal_state()		
Confirmed Fix At	N/A		

The delegator's state machine includes two states where the delegator may have an active bond: `UNBOND_NOT_ALLOWED` and `UNBOND_ALLOWED`. If a validator forcefully unbonds a delegator, the protocol allows the delegator to return to the `TERMINAL` state using the `terminal_state` transition. If the forceful unbonding happens while the delegator is in the `UNBOND_NOT_ALLOWED` state, the validator is banned:

```
if (current_state == UNBOND_NOT_ALLOWED) {
   let current_validator_state: validator_state = validator_mapping.get(0u8);
   banned_validators.set(current_validator_state.validator, true);
}
```

Snippet 4.10: Code snippet from the `finalize_terminal_state()` function.

However, this logic does not ban the validator if the delegator was forcefully unbonded while in the `UNBOND_ALLOWED`state.

Impact If a validator unbonds a delegator while the delegator is in the `UNBOND_ALLOWED` state, the validator will not be banned, which is inconsistent with the protocol's behavior when the delegator is in the `UNBOND_NOT_ALLOWED` state.

Recommendation It is advisable to ban the validator if it forcefully unbonds a delegator in the `UNBOND_ALLOWED` state as well, ensuring consistent behavior regardless of the delegator's state.

Developer Response Acknowledged

4.1.17 V-PON-VUL-017: Once banned a validator cannot be unbanned

Severity	Warning	**Commit**	8cde12d
Type	Usability Issue	**Status**	Fixed
File(s)	`pondo_delegator*.aleo`		
Location(s)	bond_failed		
Confirmed Fix At	372bba6		

The pondo oracle allows banning a validator if the validator closes itself to delegation or unbonds the pondo delegator. But, there is no way to unban a validator.

There might be legitimate reasons for a validator to close itself to delegation for a period of time. For example, if the delegator is very close to 25% of total staked aleo credits it can close itself to delegation to not cross that threshold. It can also be temporarily done to deal a delegator that keeps staking/unstaking to mess with their committee status (this is possible if the validator is very close to 10M credits.)

Even if they have a closed committee status for legitimate reasons and are high performers the validators will be banned. And they can then never be unbanned.

Impact A high performant validator who closes itself to delegation temporarily will be banned permanently.

Recommendation Consider adding a transition to be able to unban validators which meet the criteria. For security purposes this transition should only be callable by the admin.

Developer Response The developers implemented the suggested fix.

4.1.18 V-PON-VUL-018: Accidental removal of admin

Severity	Warning	Commit	8cde12d
Type	Data Validation	Status	Fixed
File(s)	pondo_oracle/main.leo		
Location(s)	remove_control_address()		
Confirmed Fix At	011e7d1		

The function `remove_control_address()` enables the admin to remove one of the control addresses. However, as shown below, there is no check that the removed control address is not the admin itself.

```
async function finalize_remove_control_address(
  public control_address: address,
  public caller: address
) {
  // Ensure the caller is an admin
  let is_admin: bool = control_addresses.get(caller);
  assert(is_admin);

  control_addresses.remove(control_address);
}
```

Snippet 4.11: Implementation of `finalize_remove_control_address()`

Additionally, the function `update_admin()` is used to update the current admin. The implementation is shown below.

```
async function finalize_update_admin(
  public new_admin: address,
  public caller: address
) {
  // Ensure the caller is an admin
  let is_admin: bool = control_addresses.get(caller);
  assert(is_admin);

  // Set the new admin
  control_addresses.set(new_admin, true);
  // Remove the old admin
  control_addresses.remove(caller);
}
```

Snippet 4.12: Implementation of `finalize_update_admin()`

Again, for this function, there is no check that the `new_admin` is not the same as the current admin (i.e., the `caller`). If they are, the last call to `control_addresses.remove(caller)` will remove both the old and new admins leaving no admin.

Impact If the admin is removed in this way, there will no longer be any address that is the admin and no way to add a new one. As a result, all admin functionalities, including critically adding delegators, will no longer be possible.

Recommendation Add a check that prevents the removal of the admin address.

Developer Response The developers implemented the suggested fix.

4.1.19 V-PON-VUL-019: Delegator allocations can sum to more than 100%

Severity	Warning	**Commit**	8cde12d
Type	Data Validation	**Status**	Acknowledged
File(s)	pondo_oracle/main.leo		
Location(s)	update_delegator_allocations()		
Confirmed Fix At	N/A		

The function `update_delegator_allocations` allows the admin to update the allocation assigned to each of the top validators. It is assumed that these values sum to `10000` (i.e., `100%`), but there is no check when these values are updated.

```
async function finalize_update_delegator_allocations(
  public multiple: [u128; 10],
  public caller: address
) {
  // Ensure the caller is an admin
  let is_admin: bool = control_addresses.get(caller);
  assert(is_admin);

  delegator_allocation.set(0u8, multiple);
}
```

Snippet 4.13: Implementation of `finalize_update_delegator_allocations()`

Impact These allocation amounts are used to determine how to split money between the various validators. If performed this way, it could lead to lost funds or potential crashes of the system when more money is allocated than is owned by the program.

Recommendation Add a check that ensures the values sum to the desired amount.

Developer Response Acknowledged

4.1.20 V-PON-VUL-020: Centralization Risk

Severity	Warning	**Commit**	8cde12d
Type	Access Control	**Status**	Fixed
File(s)	`pondo_oracle/main.leo`		
Location(s)	See issue description.		
Confirmed Fix At	cecfdbf		

The oracle in the Pondo protocol declares an administrator role that is given special permissions. In particular, these administrators are given the following abilities:

- Add and remove "control addresses" which can ban validators.
- Add new delegator allocation amounts.
- Add a proposed delegator to the delegators considered for use in the protocol.

Impact If a private key were stolen, a hacker would have access to sensitive functionality that could compromise the project. For example, a malicious admin could add a bad acting delegator that could steal funds from the protocol.

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 implemented the suggested fix.

4.1.21 V-PON-VUL-021: Lost boosting funds

Severity	Warning	**Commit**	8cde12d
Type	Logic Error	**Status**	Acknowledged
File(s)	`pondo_oracle/main.leo`		
Location(s)	boost_validator()		
Confirmed Fix At	N/A		

Boosting is used to enabled validators to add in additional funds to "boost" their ranking in the list of top performing validators. Boost is added per epoch, and is accumulated using the following logic.

```
// If the boost is in the same epoch, add the boost amount
let new_boost_amount: u64 = current_boost.epoch == current_epoch ? current_boost.
    boost_amount + boost_amount : boost_amount;
```

Snippet 4.14: Snippet from `finalisze_boost_validator()`

As shown above, if the epoch of the boost is not the current epoch, the boost amount is overwritten by the new boost amount.

Impact If a validator was not updated in the last epoch, their boost amount could be lost.

Recommendation Add a check that carries over unused boost amounts, or, at the very least, add some warning to users about the possibility of lost boost in this case.

Developer Response Acknowledged

4.1.22 V-PON-VUL-022: Unused code

Severity	Info	**Commit**	8cde12d
Type	Maintainability	**Status**	Fixed
File(s)	`pondo_core_protocol/main.leo`		
Location(s)	See description		
Confirmed Fix At	6f0191a		

1. The pondo core program declares but does not use the following constants:
   - `UNBOND_ALLOWED`
   - `UNBONDING`
   - `TERMINAL`
2. The `deposit_public_as_signer` and `deposit_public` function declares the `referrer` parameter but they do not make use of it.

Impact Unused code affects the maintainability of the project.

Recommendation Remove code that is not used.

Developer Response The developers implemented the suggested fix.

4.1.23 V-PON-VUL-023: Predefined tokens can be registered in the MTSP before pondo deployment

Severity	Info	**Commit**	8cde12d
Type	Maintainability	**Status**	Acknowledged
File(s)	`pondo_token/main.leo`		
Location(s)	See description		
Confirmed Fix At	N/A		

The pondo codebase uses predefined token IDs for `pAleo` and `pondo` tokens. This approach allows an attacker to register these token IDs in the `MTSP` contract before the pondo protocol does.

Impact The attacker will prevent the pondo protocol from deploying correctly. Additionally, an attacker could use fake tokens with these IDs in third-party contracts that assume these token IDs correspond to genuine `pAleo` and `pondo` tokens.

Recommendation It is advisable for the token id for the `pAleo` and `pondo` tokens to not be pre-defined.

Developer Response Acknowledged