Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Beta Staking

Beta Staking

Veridise Inc.
August 23, 2024

► **Prepared For:**

Beta Staking
https://betastaking.com/

► **Prepared By:**

Mark Anthony
Nicholas Brown

► **Contact Us:**

contact@veridise.com

► **Version History:**

Aug. 20, 2024	V1
Aug. 23, 2024	V2

Contents

Executive Summary 1

From Aug. 7, 2024 to Aug. 15, 2024, Beta Staking engaged Veridise to review the security of their Beta Staking program. The review covered a protocol that allows users to stake Aleo credits in exchange for a liquid stAleo token that will passively earn rewards. Veridise conducted the assessment over 14 person-days, with 2 engineers reviewing code over 7 days on commit `263c7e3d6`. The auditing strategy involved an extensive manual review of the source code performed by Veridise engineers.

Code assessment. The Beta Staking developers provided the source code of the Beta Staking contracts for review. The source code appears to be mostly original code written by the Beta Staking developers. The source code is based on the functionality of Lido Finance implemented by the Beta Staking developers.

To facilitate the Veridise auditors' understanding of the code, the Beta Staking developers met with the Veridise auditors to discuss the intended behavior of the protocol. The source code did not contain a test suite.

Summary of issues detected. The audit uncovered 7 issues, 2 of which are assessed to be of medium severity by the Veridise auditors. Specifically, V-BST-VUL-001 outlines a sandwich attack, where an attacker can stake a large amount of credits immediately before rewards are updated to the beta staking program through notify_reward and withdraw afterward to claim a large portion of the rewards being updated. V-BST-VUL-002 describes the centralization risk associated with the protocol due to the centralized nature of the onchain programs and the heavy reliance on the backend component. The Veridise auditors also identified 1 warning and 4 informational findings.

Among the 7 issues, 6 issues have been acknowledged by Beta Staking, and 1 issue has been determined to be intended behavior after discussions with Beta Staking. Of the 6 acknowledged issues, Beta Staking has fixed 4 issues and provided partial fixes to 1 more. Beta Staking plans to fix the 1 acknowledged issue through their off-chain mechanism. The 1 issue that was determined to be intended behavior has also been included in the report by the Veridise auditors, so that readers are aware of behavior which may at first seem unexpected.

Recommendations. After auditing the protocol, the auditors had a few suggestions to improve the Beta Staking. Since the behavior of the stakers is controlled by a centralized backend for the project which was out of scope of this audit, there is still a significant potential for issues when deploying this protocol. The auditors would recommend putting as much of the critical logic on chain if possible, and getting the backend implementation audited for correctness. Additionally, since this protocol is very centralized, the auditors would like to emphasize the importance of using decentralized governance or a multi-sig wallet to ensure that a stolen private key won't compromise the protocol.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
Beta Staking	`263c7e3d6`	Aleo	Aleo

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Aug. 7–Aug. 15, 2024	Manual & Tools	2	14 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Acknowledged	Fixed
Critical-Severity Issues	0	0	0
High-Severity Issues	0	0	0
Medium-Severity Issues	2	2	0
Low-Severity Issues	0	0	0
Warning-Severity Issues	1	1	1
Informational-Severity Issues	4	3	3
TOTAL	7	6	4

Table 2.4: Category Breakdown.

Name	Number
Maintainability	4
Frontrunning	1
Access Control	1
Data Validation	1

Audit Goals and Scope 3

3.1 Audit Goals

The engagement was scoped to provide a security assessment of Beta Staking's Aleo programs. In our audit, we sought to answer questions such as:

- Is it possible for a malicious user to steal funds from the protocol?
- Is it possible for a malicious user to perform a denial of service attack against the protocol?
- Does the program keep track of user balances and rewards correctly?
- Are access controls correctly enforced?
- Can a user get more rewards than they are owed?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved a manual analysis by human experts.

Scope. The scope of this audit is limited to the `beta_staking/src/main.leo` and `staker/src/main.leo`files in the source code provided by the Beta Staking developers, which contains the Aleo program implementation of the Beta Staking.

Methodology. Veridise auditors read the Beta Staking documentation and discussed the expected behavior with the developers. They then began a manual review of the code. During the audit, the Veridise auditors regularly met with the Beta Staking developers to ask questions about the code.

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:

In addition, we judge the impact of a vulnerability as follows in Table 3.3:

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

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-BST-VUL-001	The notify_reward can be sandwiched to . . .	Medium	Acknowledged
V-BST-VUL-002	Centralization Risk	Medium	Partially Fixed
V-BST-VUL-003	Missing Checks in update_settings	Warning	Fixed
V-BST-VUL-004	Unnecessary Code Duplication	Info	Intended Behavior
V-BST-VUL-005	ZERO_ADDR constant is defined but not used	Info	Fixed
V-BST-VUL-006	Incorrect Comment	Info	Fixed
V-BST-VUL-007	Unnecessary Hardcoded Constants	Info	Fixed

4.1 Detailed Description of Issues

4.1.1 V-BST-VUL-001: The notify_reward can be sandwiched to get a large portion of rewards

Severity	Medium	**Commit**	263c7e3
Type	Frontrunning	**Status**	Acknowledged
File(s)	beta_staking/src/main.leo		
Location(s)	notify_reward		
Confirmed Fix At	N/A		

The staking rewards in the `beta_staking` program are updated when a staker calls `notify_reward` with a specific reward amount. See snippet below for context.

An attacker can deposit a large amount of Aleo credits just before a `notify_reward` transition. They can then unstake immediately afterwards and take away a large portion of the rewards being added into the pool reserves. There are no fees for staking/unstaking and the protocol only takes a fee percentage from the rewards. Therefore, the process described above can be repeated for each rewards update cycle without incurring a substantial fee cost.

In this particular scenario, because the user's credits are never actually bonded, they should not be eligible for any of the staking rewards. But they do get part of the rewards without any assurance that their funds were indeed staked.

So, essentially `notify_reward` can be sandwiched between `stake_public` and `unstake_public` to gain a portion of staking rewards which the user does not deserve.

```
// Notify staking rewards of the pool
async transition notify_reward(public aleo_amount: u64) -> Future {
    assert_neq(aleo_amount, 0u64);
    return finalize_notify_reward(self.caller, aleo_amount);
}
```

Snippet 4.1: Snippet from `notify_reward()`

Impact An attacker can deposit large amounts of Aleo credits just before a staker updates rewards through `notify_reward` and unstake immediately afterwards to take away a large portion of the rewards being updated.

Recommendation The rewards being updated in `notify_reward` should be small enough to ensure that repeated cycles of the above scenario are unprofitable for the attacker. Thus `max_reward_per_notify` should be set to a small value relative to the `total_reserve`.

Developer Response The backend will try to call notify_reward as often as possible, while minimizing the amount of rewards in a single notify.

In the meantime, the monitoring service will monitor transactions and program states specifically for this issue.

4.1.2 V-BST-VUL-002: Centralization Risk

Severity	Medium	**Commit**	263c7e3
Type	Access Control	**Status**	Partially Fixed
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	cd8a043		

Similar to many projects, Alphaswap's, `beta_staking` and `staker` program's declare an administrator role that is given special permissions. The `staker` program also contains an `operator` role. In particular, these administrators are given the following abilities:

- `beta_staking`
  - An admin can update key settings for the `beta_staking` program like the account that receives the fees and the fee amounts.
  - An admin can register `stakers` in the `beta_staking` program which have permission to pull funds from the program
  - An admin can withdraw arbitrary funds from the `beta_staking` protocol.
- `staker`
  - An `operator` can control whether funds are staked/unstaked and can pull and push funds between the `staker` and the `beta_staking` program
  - An `admin` can register new operators for the `staker` program

Additionally, the `beta_staking` program relies heavily on these `stakers` behaving in the best interest of the protocol. The developers have indicated that the stakers will be controlled by a backend that is out of scope for this audit. For the rest of the issues we have assumed that the backend behaves perfectly. However, a centralized, off-chain backend can lead to major issues if there are implementation bugs or downtime.

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` for the `beta_staking` program could steal arbitrary funds from the program. A malicious `admin` or `operator` of a registered `staker` could perform a denial of service attack by unstaking funds and refusing to update the rewards.

A centralized backend could also lead to issues including, but not limited to the following:

- If the backend goes offline, user funds that are being unstaked will be frozen if the stakers haven't already transferred the necessary funds to the `beta_staking` funds. Unstaked funds don't earn rewards so this could also result in monetary losses (in addition to not being able to access their funds) for users waiting for the backend to come online.
- If the backend doesn't direct the stakers to send sufficient funds to the `beta_staking` protocol to cover all unstaking requests, starvation could occur for slower users if many users have had their waiting period elapse.
- Conversely if the backend leaves too many tokens in the `beta_staking` protocol, this could lead to inefficiencies and lower rewards for users.

Recommendation

- The operations mentioned in the description above are all particularly sensitive operations. We would encourage the developers to utilize a decentralized governance or multi-sig contract for privileged roles as opposed to a single account, which introduces a single point of failure.
- The amount of funds that the individual stakers can pull to themselves should be restricted to a certain percentage of the total funds in the beta staking program. This way the amount of funds that are supposed to go to individual stakers will be clearly documented. This can provide more transparency to protocol fund management.
- The off-chain backend should also be audited for correctness, or the stakers should be managed through on-chain functionality.

Developer Response The strategies we've adopted:

- The backend will only have operator privileges, it can only perform staking related operations, and does not have more advanced administrative privileges.
- To prevent the backend from getting bugs that could lead to a reduction in staking rewards, we have a monitoring service that monitors the behaviour of the backend and program states, which will alert us as soon as the backend is not working correctly.
- To prevent malicious stakers from being added for any reason in the future, we have added a new setting field - `fixed_stakers`. We will deploy and add all necessary stakers after the staking program has been deployed, and then we will turn on the setting to disable the addition of any new stakers.

We'll continue to implement these recommendations in the future:

- Utilize a decentralized governance or multi-sig contract for privileged roles.
- The off-chain backend should also be audited for correctness, or the stakers should be managed through on-chain functionality.

4.1.3 V-BST-VUL-003: Missing Checks in update_settings

Severity	Warning	**Commit**	263c7e3
Type	Data Validation	**Status**	Fixed
File(s)	`beta_staking/src/main.leo`		
Location(s)	finalize_update_settings()		
Confirmed Fix At	cd8a043		

The `finalize_update_settings()` function performs a check on the new settings. However this only checks the value of the `protocol_fee`.

```
    async function finalize_update_settings(public caller: address, public s:
    Settings) {
        // Only admins can perform this action
        assert_eq(admins.get(caller), true);
        // Check settings
        assert(s.protocol_fee <= 10000u16);

        // Update settings
        settings.set(true, s);
    }
```

Snippet 4.2: Definition of `finalize_update_settings()`

Impact Other fields in the updated settings could be incorrectly set to an unreasonable value which could cause problems for the protocol.

Recommendation Add checks for the following fields:

- `wait_time`: should at least check that this value is greater than 0
- `max_reward_per_notify`: should check that this value is greater than 0 (or some other sane minimum)

4.1.4 V-BST-VUL-004: Unnecessary Code Duplication

Severity	Info	**Commit**	263c7e3
Type	Maintainability	**Status**	Intended Behavior
File(s)	`staker/src/main.leo`		
Location(s)	claim(), push_aleo(), claim_and_push()		
Confirmed Fix At	N/A		

The `staker.aleo` program implements a `claim()` function that allows an operator to claim the unbonding microcredits and a `push_aleo()` function that will send credits back to the `beta_staking.aleo` program. Additionally the `staker.aleo` program implements a `claim_and_push()` that has an identical effect as calling `claim()` then calling `push_aleo()`

```
// Claim unbonding microcredits
    async transition claim() -> Future {
        let f1: Future = credits.aleo/claim_unbond_public(self.address);
        return finalize_claim(self.caller, f1);
    }

    async function finalize_claim(public caller: address, public f1: Future) {
        f1.await();
        // Only operators can perform this action
        assert_eq(operators.get(caller), true);
    }
```

Snippet 4.3: Definition of `claim()`

```
    async transition push_aleo(public push_amount: u64) -> Future {
        assert_neq(push_amount, 0u64);
        let f1: Future = credits.aleo/transfer_public(beta_staking.aleo, push_amount)
    ;
        return finalize_push_aleo(self.caller, push_amount, f1);
    }

    async function finalize_push_aleo(public caller: address, public push_amount: u64
    , public f1: Future) {
        f1.await();
        // Only operators can perform this action
        assert_eq(operators.get(caller), true);
        // Update states
        let pre_state: StakerState = state.get(true);
        state.set(true, StakerState {
            total_pulled: pre_state.total_pulled,
            total_pushed: pre_state.total_pushed + push_amount,
            total_reward: pre_state.total_reward,
        });
    }
```

Snippet 4.4: Definition of `push_aleo()`

Impact Since code is duplicated between the two functions there could be issues in the future if the implementation needs to change and the functions aren't properly kept in sync.

```
    // Claim unbonding microcredits
    async transition claim_and_push(public push_amount: u64) -> Future {
        let f1: Future = credits.aleo/claim_unbond_public(self.address);
        let f2: Future = credits.aleo/transfer_public(beta_staking.aleo, push_amount)
    ;
        return finalize_claim_and_push(self.caller, push_amount, f1, f2);
    }

    async function finalize_claim_and_push(public caller: address, public push_amount
    : u64, public f1: Future, public f2: Future) {
        f1.await();
        f2.await();
        // Only operators can perform this action
        assert_eq(operators.get(caller), true);
        // Update states
        let pre_state: StakerState = state.get(true);
        state.set(true, StakerState {
            total_pulled: pre_state.total_pulled,
            total_pushed: pre_state.total_pushed + push_amount,
            total_reward: pre_state.total_reward,
        });
    }
```

Snippet 4.5: Definition of `claim_and_push()`

Recommendation Remove the `claim_and_push()` function.

Developer Response Combining the two operations in claim_and_push() simplifies the backend implementation, so this interface will be retained.

4.1.5 V-BST-VUL-005: ZERO_ADDR constant is defined but not used

Severity	Info	**Commit**	263c7e3
Type	Maintainability	**Status**	Fixed
File(s)	`staker/src/main.leo`		
Location(s)	See description		
Confirmed Fix At	cd8a043		

The `ZERO_ADDR` constant is defined in the `beta_staking` program but it isn't used. However, a hard coded zero address with an identical value is used later in the program.

```
import credits.aleo;
import beta_staking.aleo;

// The 'staker' program.
program staker.aleo {
const DEFAULT_ADMIN: address =
    aleo18jfrqsz4m853grpgzflrlzdkcm9art926668g80vd9ruyzv8rsqqlchzyj;
// The constant below
const ZERO_ADDR: address =
    aleo1qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq3ljyzc;
```

Snippet 4.6: Snippet from `beta_staking`

```
let bond: bond_state = Mapping::get_or_use(credits.aleo/bonded, self.address,
    bond_state {
    validator: aleo1qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq3ljyzc,
    microcredits: 0u64
});
```

Snippet 4.7: Snippet from `finalize_notify_reward()`

Impact This hurts maintainability and could lead to inconsistent values for the zero address if this constant is used again in the code.

Recommendation Use the `ZERO_ADDR` constant instead of hard coding the value.

4.1.6 V-BST-VUL-006: Incorrect Comment

Severity	Info	**Commit**	263c7e3
Type	Maintainability	**Status**	Fixed
File(s)	`beta_staking/src/main.leo`		
Location(s)	see issue description		
Confirmed Fix At	cd8a043		

The `Settings` Struct in the `beta_staking` program describes how the protocol fee is 1000 by default and indicates that this value is a 0.1% fee. Based on how this fee is used it appears to be measured in basis points (10000 basis points = 100%). Thus 1000 basis points will be 10%, not 0.1%

```
struct Settings {
    // Waiting blocks to withdraw after unstake
    unstake_wait: u32,
    // The flag used to pause stake function
    stake_paused: bool,
    // The flag used to pause some functions in emergency situations
    global_paused: bool,
    // Max reward can be notified in one time
    max_reward_per_notify: u64,
    // Protocol fee, default is 1000 (0.1%)
    protocol_fee: u16,
    fee_account: address,
}
```

Snippet 4.8: Definition of the `Settings` struct

Impact This comment could lead to developer confusion when updating the `protocol_fee` which could lead the admin to update the `protocol_fee` to 10000 thinking that the new fee is 1%. However, such a situation would result in the protocol taking 100% of the rewards.

Recommendation Update the comment to reflect that the default fee of 1000 corresponds to 10%.

4.1.7 V-BST-VUL-007: Unnecessary Hardcoded Constants

Severity	Info	**Commit**	263c7e3
Type	Maintainability	**Status**	Fixed
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	cd8a043		

In several places in the codebase, there are hardcoded numerical constants used. These values can be managed more easily as Leo constants, and if they are managed in multiple places, using a Leo constant will ensure that the values are the same throughout the codebase.

Hardcoded numerical constants are used in the following places:

- `beta_staking.aleo`
  - `finalize_notify_reward()` lines 419-420
  - `finalize_update_settings()` line 456
    - `10000` is used based on the units of the `protocol_fee`

Impact For the hardcoded value `10000`, since it is used in so many places, using a Leo constant instead could prevent a typo (like an extra 0) from occurring in one of the locations in which it is used.

Recommendation Use Leo constants instead of hardcoded numerical constants.

Glossary

Lido Finance Lido finance is a liquid staking solution focused on Ethereum. To learn more, visit `https://docs.lido.fi/`. 1