Composable Finance

Security Assessment

February 14th, 2024 — Prepared by OtterSec

Akash Gurugunti sud0u53r.ak@osec.io

Robert Chen r@osec.io

Table of Contents

Appendices

01 — Executive Summary

Overview

Composable Finance engaged OtterSec to assess the `emulated-light-client` program. This assessment was conducted between January 16th and January 19th, 2024. For more information on our auditing methodology, refer to Appendix B.

Key Findings

We produced 10 findings throughout this audit engagement.

In particular, we identified a critical vulnerability concerning unauthorized alterations to staking parameters (OS-CFI-ADV-00) and another issue regarding the lack of account validation in the deposit and staking functionalities (OS-CFI-ADV-01). Furthermore, we highlighted a potential lockup of funds during withdrawal due to a lack of compatibility for handling cases where no passing of a value to the optional service parameter occurs (OS-CFI-ADV-05).

We also made recommendations around optimizing token transfer and burning non-fungible tokens by introducing a boolean argument or utilizing empty seeds to enable unsigned invocation where signed invocation is unnecessary (OS-CFI-SUG-03) and proposed the replacement of a method in the set state function (OS-CFI-SUG-00). Additionally, we advised the inclusion of certain constraints in the withdraw instruction for the staking params account (OS-CFI-SUG-02).

02 — Scope

The source code was delivered to us in a Git repository at https://github.com/ComposableFi/emulated-light-client. This audit was performed against commit aaf20a2.

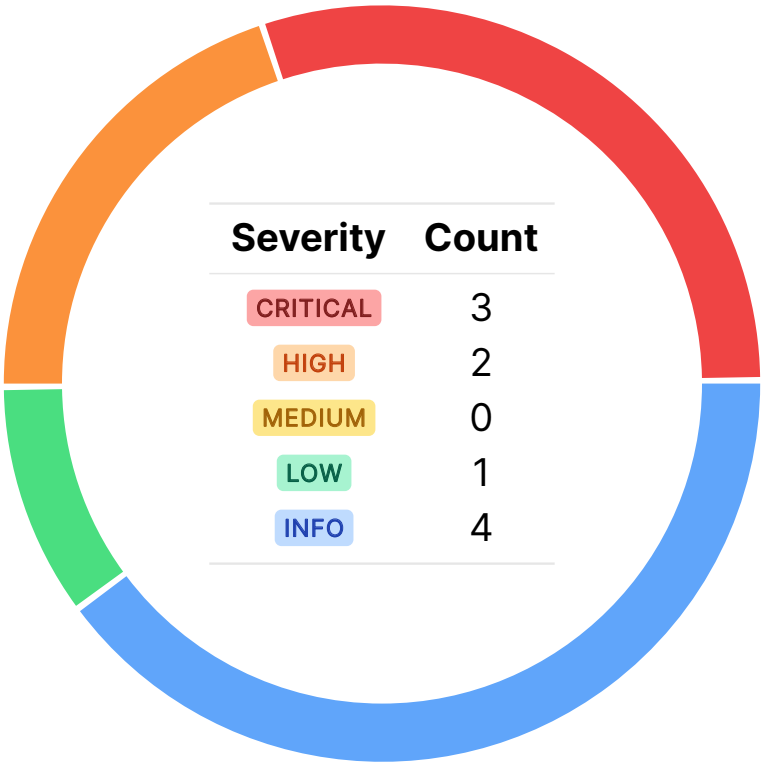
A brief description of the programs is as follows:

Name	Description
emulated-light-client	A program that describes a bridge between Solana and Cosmos utilizing inter-blockchain communication.

03 — Findings

Overall, we reported 10 findings.

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



Severity	Count
CRITICAL	3
HIGH	2
MEDIUM	0
LOW	1
INFO	4

04 — Vulnerabilities

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

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-CFI-ADV-00	CRITICAL	RESOLVED	`Initialize` utilizes `init_if_needed`, allowing unauthorized alterations to staking parameters.
OS-CFI-ADV-01	CRITICAL	RESOLVED	Lack of account validation for `remaining_accounts`
OS-CFI-ADV-02	CRITICAL	RESOLVED	Absence of a check for a non-zero balance in the depositor's `receipt_token_account` within `set_service` instruction.
OS-CFI-ADV-03	HIGH	RESOLVED	Missing parameters in `set_stakesolana_ibc::cpi::set_stake` function, for identifying the specific mint of the staked amount.
OS-CFI-ADV-04	HIGH	RESOLVED	Lack of validation for the instruction sysvar account in `validate_remaining_accounts` and `set_stake`, potentially allowing unintended or insecure usage.
OS-CFI-ADV-05	LOW	RESOLVED	Potential lockup of funds during withdrawal due to a lack of compatibility with `None` values for the optional `service` parameter.

Ability To Initialize Multiple Times CRITICAL OS-CFI-ADV-00

Description

In `Initialize`, while initializing the staking parameters, due to `init_if_needed`, anyone may alter the staking parameters with new values multiple times. The ability to initialize the staking parameters multiple times may result in potential security vulnerabilities. For example, an attacker could repeatedly call `Initialize` with different parameters, altering the staking configuration and affecting the entire protocol.

Remediation

Utilize `init` instead of `init_if_needed` for `Initialize`. This ensures that the initialization only occurs once.

Patch

Fixed by using `init` instead of `init_if_needed` for `Initialize` in e565006.

Discrepancy In Deposit Functionality CRITICAL OS-CFI-ADV-01

Description

`deposit` utilizes `remaining_accounts` for the cross-program invocation call to the guest chain program (`solana_ibc::cpi::set_stake`). However, the function lacks explicit validation checks on `remaining_accounts` . `solana_ibc::cpi::set_stake` is the invoked cross-program invocation call. Similarly, this function also lacks explicit validation checks for the accounts passed in `CpiContext` .

>_ *restaking/programs/restaking/src/lib.rs* rust

```rust
pub fn deposit<'a, 'info>(
    ctx: Context<'a, 'a, 'a, 'info, Deposit<'info>>,
    service: Option<Service>,
    amount: u64,
) -> Result<()> {
    [...]
    // Call Guest chain program to update the stake if the chain is initialized
    if guest_chain_program_id.is_some() {
        [...]
        let cpi_program = ctx.remaining_accounts[3].clone();
        let cpi_ctx =
            CpiContext::new_with_signer(cpi_program, cpi_accounts, seeds);
        solana_ibc::cpi::set_stake(cpi_ctx, amount as u128)?;
    }
    Ok(())
}
```

Remediation

Add validation checks in both `deposit` and `solana_ibc::cpi::set_stake` to ensure that the required accounts are present and have the correct ownership.

Patch

Fixed by adding validation checks to the `remaining_accounts` in b7847d9.

Missing Receipt Token Balance Check CRITICAL OS-CFI-ADV-02

Description

In the implementation of `set_service` instruction, there is a section of code that sets the service for the stake which was deposited before guest chain initialization without explicitly checking if the depositor's `receipt_token_account` has a non-zero balance. The code assumes that the depositor has a sufficient balance in their `receipt_token_account` to cover the stake, but it fails to check for it explicitly.

>_ *restaking/programs/restaking/src/lib.rs* rust

```rust
pub fn set_service<'a, 'info>(
    ctx: Context<'a, 'a, 'a, 'info, SetService<'info>>,
    service: Service,
) -> Result<()> {
    [...]
    vault_params.service = Some(service);
    let guest_chain_program_id =
        staking_params.guest_chain_program_id.unwrap(); // Infallible
    let amount = vault_params.stake_amount;
    [...]
}
```

Proof of Concept

- A malicious user sets an arbitrary service for a genuine user's `vault_params` by calling the `set_service` instruction using the genuine user's `vault_params` and an arbitrary `Service`.
- Since the code does not check for a non-zero balance in the `receipt_token_account`, the malicious user may abuse the system by setting an unauthorized stake using the original depositor's `vault_params`.

Remediation

Explicitly check if the depositor's `receipt_token_account` has a non-zero balance before proceeding with the stake setting. This check ensures the depositor has access to the respective `vault_params`.

Patch

Fixed by checking if the depositor's `receipt_token_account` has a non-zero balance in e10222d.

Stake Mint Differentiation HIGH OS-CFI-ADV-03

Description

The vulnerability is rooted in `deposit`, and pertains to the lack of consideration for different token decimals in `solana_ibc::cpi::set_stake`, specifically related to the mint of the staked amount. `set_stake` is invoked without explicitly providing information about the mint of the staked amount.

>_ *restaking/programs/restaking/src/lib.rs* rust

```rust
pub fn deposit<'a, 'info>(
    ctx: Context<'a, 'a, 'a, 'info, Deposit<'info>>,
    service: Option<Service>,
    amount: u64,
) -> Result<()> {
    [...]
    // Call Guest chain program to update the stake if the chain is initialized
    if guest_chain_program_id.is_some() {
        [...]
        let cpi_program = ctx.remaining_accounts[3].clone();
        let cpi_ctx =
            CpiContext::new_with_signer(cpi_program, cpi_accounts, seeds);
        solana_ibc::cpi::set_stake(cpi_ctx, amount as u128)?;
    }
    Ok(())
}
```

On invoking `solana_ibc::cpi::set_stake`, it's crucial to include parameters that identify the specific mint of the staked amount. Tokens on solana may have different decimal places, and each mint may have a different scale. Without passing information about the mint of the staked amount, there's a risk of updating the stake value with an incorrect scale. For instance, staking 10 tokens with 2 decimals and staking 10 tokens with 6 decimals may result in updating the same value, thus causing the staked amount to be incorrectly represented.

Remediation

Modify `set_stake` to accept parameters explicitly indicating the mint of the staked amount. This adjustment ensures the program accurately manages and updates stakes for various token types.

Patch

Fixed by adding a check in `deposit` instruction to check if the `token_mint` has only 9 decimals in d78a19a.

Lack Of Sysvar Account Validation HIGH OS-CFI-ADV-04

Description

The program passes the instruction sysvar account to both `deposit` and `set_service` instructions, but does not perform validation in the `validate_remaining_accounts` and even in the `solana_ibc::cpi::set_stake` function. Thus, replacing the instruction sysvar account is possible. They might be able to inject unauthorized instructions into the cross-program invocation calls.

Remediation

Ensure both `validation::validate_remaining_accounts` and `solana_ibc::cpi::set_stake` include explicit validation for the instruction sysvar account.

Patch

Fixed by checking the instruction sysvar account in b221448.

Potential Fund Lockup LOW OS-CFI-ADV-05

Description

`deposit` includes an optional `service` parameter, which is of type `Option<Service>`. This parameter specifies a service associated with the staking operation. The vulnerability arises from the presence of the service parameter, which is a condition during withdrawal. Specifically, the withdrawal logic includes a check on the service parameter.

>_ *restaking/programs/restaking/src/lib.rs* rust

```rust
pub fn deposit<'a, 'info>(
    ctx: Context<'a, 'a, 'a, 'info, Deposit<'info>>,
    service: Option<Service>,
    amount: u64,
) -> Result<()> {
    [...]
     vault_params.service =
            if guest_chain_program_id.is_some() { service } else { None };
    [...]
}
```

The code assumes that the service parameter will always be `Some(service)` during withdrawal. However, calling `deposit` with `None` for the service parameter would not fulfill the condition. Thus, If a deposit occurs without specifying a service (i.e., passing `None`), and the withdrawal logic assumes that there is always a service (`is_some()` condition), it may lock up funds.

Remediation

Modify `withdraw` to ensure compatibility with `None` values for the service parameter preventing fund lockup.

Patch

Fixed by added an additional instruction `set_service` to set the `service` parameter after depositing funds in 57edfe8.

05 — General Findings

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

ID	Description
OS-CFI-SUG-00	Proposal to replace `CpiContext::new_with_signer` with `CpiContext::new` in `deposit` and `set_service` instructions.
OS-CFI-SUG-01	Change the `Option<Service>` parameter to `Service` in `deposit` to prevent users from setting `vault_params.service` to `None`.
OS-CFI-SUG-02	`Withdraw` does not include the `has_one = rewards_token_mint` constraint for the `staking_params` account.
OS-CFI-SUG-03	Optimizing `token::transfer` and `burn_nft` by introducing a boolean argument or utilizing empty seeds to enable unsigned invocation when signed invocation is unnecessary.

Context Signer Correction OS-CFI-SUG-00

Description

`CpiContext::new_with_signer` creates the context for cross-program invocation. This method is typically used when a program expects a signed invocation, including the account's seeds for signature verification.

Remediation

In `deposit` and `set_service` instructions , utilizing `CpiContext::new` to call `set_stake` is more appropriate and simplifies the context creation.

Enforce Mandatory Service Assignment OS-CFI-SUG-01

Description

While handling the `vault_params.service` field in `deposit`, currently, it is defined as `Option<Service>`, allowing it to be either `Some(Service)` or `None`. The logic in `deposit` utilizes this option to conditionally set the `vault_params.service` based on `guest_chain_program_id.is_some`.

>_ *restaking/programs/restaking/src/lib.rs* rust

```rust
pub fn deposit<'a, 'info>(
    ctx: Context<'a, 'a, 'a, 'info, Deposit<'info>>,
    service: Option<Service>,
    amount: u64,
) -> Result<()> {
    [...]
}
```

However, the issue arises from the fact that even if `guest_chain_program_id` is `Some`, indicating the guest chain is initialized, the service may still be set to `None`. This is due to the optionality of the `Service` type. Users may set `vault_params.service` to `None` even after initializing the chain, which could undermine the intended behavior of the logic.

Remediation

Change the type of `vault_params.service` from `Option<Service>` to just `Service`. This modification ensures that a valid Service must always be provided once the guest chain is initialized.

Missing Constraint

OS-CFI-SUG-02 *

Description

In `Claim`, there is a `has_one` constraint for the `rewards_token_mint` field in the `staking_params` account. This constraint ensures a one-to-one relationship between the `staking_params` account and the associated `rewards_token_mint`. However, `Withdraw` lacks this constraint.

Remediation

Extend this constraint to `Withdraw`, to maintain the consistency in the code base.

Code Optimization OS-CFI-SUG-03

Description

There is unnecessary signing and seed usage in certain `token::transfer` and `burn_nft` function calls. Specifically, in `deposit`, calls to `token::transfer` do not require seeds or a signed invocation. In `Withdraw`, there is a call to `burn_nft`, which also does not require a signed invocation—resulting in unnecessary signing and seed usage affecting the code's efficiency.

Remediation

Modify `token::transfer` and `burn_nft` to handle both signed and unsigned invocations. This may be achieved by introducing a boolean argument or checking the seeds' length.

A — Vulnerability Rating Scale

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

CRITICAL

Vulnerabilities that immediately result in a loss of user funds with minimal preconditions.

Examples:

- Misconfigured authority or access control validation.
- Improperly designed economic incentives leading to loss of funds.

HIGH

Vulnerabilities that may result in a loss of user funds but are potentially difficult to exploit.

Examples:

- Loss of funds requiring specific victim interactions.
- Exploitation involving high capital requirement with respect to payout.

MEDIUM

Vulnerabilities that may result in denial of service scenarios or degraded usability.

Examples:

- Computational limit exhaustion through malicious input.
- Forced exceptions in the normal user flow.

LOW

Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk.

Examples:

- Oracle manipulation with large capital requirements and multiple transactions.

INFO

Best practices to mitigate future security risks. These are classified as general findings.

Examples:

- Explicit assertion of critical internal invariants.
- Improved input validation.

B — Procedure

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

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

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

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

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

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

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