Code Audit

for

Symmetry Protocol

https://opcodes.fr / contact@opcodes.fr

15 Rue des Halles, 75001 Paris, France

Project Information

Project	
Mission	Code audit
Client	Symmetry protocol
Start Date	03/12/2022
End Date	03/19/2022

Document Revision			
Version	Date	Details	Authors
1.0	03/18/2022	Document creation	Thibault MARBOUD Xavier BRUNI
1.1	03/19/2022	Peer review	Baptiste OUERIAGLI
2.0	03/20/2022	Public version	Xavier BRUNI

Table of Contents

Overview

Mission Context

The purpose of the mission was to perform a code audit to discover issues and vulnerabilities in the mission scope. Comprehensive testing has been performed utilizing automated and manual testing techniques.

Mission Scope

As defined with Symmetry Protocol before the mission, the scope of this assessment was a staking Solana program. The code source was supplied through a GitHub repository:

- https://github.com/symmetry-protocol/staking / 35f3b73 (main)

OPCODES engineers were due to strictly respect the perimeter agreed with Symmetry Protocol as well as respect ethical hacking behavior.

At the end of the audit, Symmetry patched all the reported vulnerabilities. These patches have been reviewed by OPCODES up to commit 95dc866. For each vulnerability, its write-up contains a link to the resolution commit.

Note: OPCODES engineers audited the main branch of the staking repository

Project Summary

Symmetry is a decentralized index protocol and portfolio management platform on Solana. They have built a staking program where users can lock tokens in vaults and receive rewards based on their weight in it. Their weight is derived from the number of tokens they are locking and the time they are locking them for. As they lock in for a longer period, coefficients defined by the pool authority increase the stakers weight. A vault can contain multiple reward pools which allows users to be rewarded by multiple token distribution models for a single deposit.

One of the use cases of the staking program for Symmetry is to distribute fees collected from their automated market maker aggregator. Each week, they will swap collected fees for PRISM tokens and supply them to the staking reward pool. The tokens then get evenly distributed between the stakers over the course of the next week. Additionally, PRISM tokens will also be distributed linearly on a longer period so that some rewards are independent of the aggregator volume and profit.

The application is developed with Anchor framework v0.22 and perfectly leverages its accounts structures and security constraint capabilities. This also allows for easier testing and frontend integration using the tools provided by Anchor ecosystem.

Synthesis

Security Level: GOOD

The overall security level is considered as good.

OPCODES investigation produced 1 major severity result, it concerned exploitable overflows that were present in multiple places throughout the code. If exploited, this vulnerability could have an important impact on the vault, allowing a malicious user to tamper with its balance.

One medium vulnerability regarded the fact that the close authority of the reward account was not checked in vault initialization and could cause users’ tokens to be locked forever in it.

All the vulnerabilities have been patched at the end of the audit.

Vulnerabilities summary

Total vulnerabilities	3
🟥 Critical	0
🟧 Major	1
🟨 Medium	1
🟩 Minor	1
🟦 Informational	0

Vulnerabilities & issues table

Identified vulnerabilities

Ref	Vulnerability title	Severity	Remediation effort	Status
#1	Multiple exploitable integer overflows	🟧 Major	🟨 Medium	🟩 Fixed
#2	Missing close authority check on reward token account	🟨 Medium	🟩 Low	🟩 Fixed
#3	Missing mint checks on vault token accounts	🟩 Minor	🟩 Low	🟩 Fixed

Identified vulnerabilities

Multiple exploitable integer overflows

Severity	Remediation effort	Status
🟧 Major	🟨 Medium	🟩 Fixed

Description

The program uses unsafe arithmetic that can lead to integer overflow/underflow.

Multiple exploitable overflows are present in the code. For example, in the stake instruction, it is possible to pass a large *lock_time* to get the best coefficient and maximize the *weighted_tvl* while overflowing *user_state.locked_until*. This will give the possibility for the user to unstake at any time.

In the following example, OPCODES engineers pushed coefficients up to four, for a locking period of more than one year.

```
coeffs.push(new BN(0), new BN(1), new BN(1));    // timestamp numerator denominator
coeffs.push(new BN(86400), new BN(2), new BN(1)); // if users locks for at least 1 day coefficient would be 2
coeffs.push(new BN(2628000), new BN(3), new BN(1)); // if users locks for at least 1 month coefficient would be 3
coeffs.push(new BN(31536000), new BN(4), new BN(1)); // if users locks for at least 1 year coefficient would be 4
```

Then, staked tokens and locked them for 480 years, so that: *(timestamp+lock_time) * 1_000_000_000 > u64::MAX*

```
tests.overflow && it("Example overflow", async () => {
const u64MAX = 18446744073709551615;
await stakeForUser(stakingInfo, userA, 100 * 10 ** 6, Math.floor(u64MAX / 10 ** 9 - Date.now() / 1000) + 10); //
userA locks 100 tokens with lock period = +- 480 years
console.log("userA weighted tvl", (await (await
program.account.userState.fetch(userA.userState.publicKey)).weightedTvl).toNumber()); // userA weighted tvl
400000000
console.log("\nToken locked until : ", new Date((await (await
program.account.userState.fetch(userA.userState.publicKey)).lockedUntil).toNumber() / 10 ** 9));
let balanceBeforeUnstake = (await connection.getTokenAccountBalance(userA.userTokenAccount)).value.uiAmount;
console.log("\nToken balance : ", balanceBeforeUnstake);
await unstakeForUser(stakingInfo, userA);
let balanceAfterUnstake = (await connection.getTokenAccountBalance(userA.userTokenAccount)).value.uiAmount;
console.log("\nToken balance after immediate unstake :", balanceAfterUnstake);
});
```

You will notice that the user's TVL got weighted with the maximum coefficient (4x) and that the tokens are locked until 1970 meaning they that are withdrawable immediately.

```
STAKE 100 at 1647684746 for 16799059337 seconds user: CfWvnNkF9EfzYhBfoxJeEmJvEUwmFLRLR8fGk6LuQzPZ
userA weighted tvl 400000000

Token locked until :  1970-01-01T00:00:00.008Z

Token balance :  9900
UNSTAKE at 1647684747 user: CfWvnNkF9EfzYhBfoxJeEmJvEUwmFLRLR8fGk6LuQzPZ

Token balance after immediate unstake : 10000
```

Other overflows are exploitable in the program and could have a major impact like for the calculations of the TVL and weighted TVL.

```
vault_state.tvl = vault_state.tvl + user_state.tvl;
vault_state.weighted_tvl = vault_state.weighted_tvl + user_state.weighted_tvl;
```

Scope

Staking program

Risk

An attacker might use overflows to tamper with his weighted TVL and the lock time of its tokens. With a high portion of supply and malicious user could exploit this issue and drain a part of the reward pool.

Remediation

We recommend turning on overflow checks in release mode by setting `overflow-checks = true` under `[profile.release]` in the `Cargo.toml` at the workspace root.

`staking/Cargo.toml`

```toml
[profile.release]
overflow-checks = true

[workspace]
members = [
    "programs/*"
]
```

Symmetry should also reassess all the calculations that might overflow and try to handle them case by case. If overflow-checks prevent overflows and critical scenarios, it makes the program panic and does not offer any way to handle the error. It should only be used as last-ditch safeguard.

Fix

The issue has been fixed with commit [3959ce7]() at the end of the audit.

Symmetry followed OPCODES recommendation to add *`overflow-checks = true`*.

In addition to that, the *`weighted_tvl`* for both the vault and the user state was changed to *`u128`*. Constraints were added to limit the *`lock_time`* and coefficient values.

Missing close authority check on reward token account

Severity	Remediation effort	Status
🟧 Medium	🟩 Low	🟩 Fixed

Description

The close authorities of the token accounts are not checked in the *CreateVault* instruction.

A malicious user could open a vault with a *pda_rewards_token_account* for which he is the *close_authority* and the *pda_account* is the owner. The close authority of a *TokenAccount* can close the account if its balance is zero. This remove all lamports and delete the *TokenAccount*. If this scenario happens, all the users that staked into the vault will become unable to withdraw as the *Unstake* instruction will fail to verify the integrity of the *TokenAccount* because it doesn't exist anymore. It will result in stakers tokens being locked into the *pda_locked_token_account*.

Scope

Staking program

Risk

OPCODES engineers identified the following probable scenario:

- Eve creates 2 token accounts that she owns and 1 *VaultState* account
- Eve sets the close authorities of the accounts to herself
- Eve transfers ownership of both token accounts to the vault PDA account (here, the close authority does not get removed !!)
- Eve initializes a new vault with the 2 token accounts. All constraints pass.
- Eve advertises a high APY for the vault and state that rewards distribution will start the next day
- Users start staking into the vault

- Eve never calls *SupplyRewardPool* and closes the *pda_rewards_token_account*
- **Users are unable to unstake**

Remediation

OPCODES engineers recommend creating the token accounts inside the *CreateVault* instruction. As showed below, it can be done easily with Anchor framework. As the token accounts will be new, the close authorities will be empty.

```
#[account(seeds = [vault_state.to_account_info().key.as_ref()], bump)]
pub pda_account: AccountInfo<'info>,
#[account(
    init,
    payer = authority,
    token::mint = mint,
    token::authority = pda_account,
)]
pub pda_locked_token_account: Account<'info, TokenAccount>,
#[account(
    init,
    payer = authority,
    token::mint = mint,
    token::authority = pda_account,
)]
pub pda_rewards_token_account: Account<'info, TokenAccount>,
pub mint: Account<'info, Mint>,
```

Alternatively, the token authority can be checked by a constraint. Anchor is also planning to forbid by default *TokenAccount*s with a close authority.

Fix

The issue has been fixed with commit 3959ce7 at the end of the audit.

Symmetry followed OPCODES recommendation to create the token accounts inside the create vault instruction via anchor and CPI.

Missing mint checks on vault token accounts

Severity	Remediation effort	Status
🟩 Minor	🟩 Low	🟩 Fixed

Description

Tokens are separated in 2 token accounts inside the vault:

- pda_locked_token_account: contains all the locked tokens of the stakers
- pda_rewards_token_account: contains the rewards to be distributed to the stakers

The *create_vault* instruction does not check that the mint address of these 2 token accounts are the same. The locked and rewards token account must have the same mint because the instructions *add_stake* and *compound* transfer tokens between them. If the mints are not the same, the instruction will fail as the spl token program error out.

Scope

Staking program

Risk

A user cannot call *add_stake* if he is already owed some rewards. Moreover, the program hasn't been built to support different currencies for locked and reward token accounts and weird behavior could surface in the future if instructions were to be added or modified.

Remediation

Creating the token accounts via anchor as defined in #2 will prevent mints from being different.

Fix

The issue has been fixed with commit 3959ce7 at the end of the audit.

Symmetry followed OPCODES recommendation to create the token accounts inside the create vault instruction via anchor and CPI.

Conclusion

Symmetry staking program is well commented and has good testing coverage. It leverages Anchor framework efficiently as well as its testing toolset which made it easy to test the vulnerabilities uncovered.

The team was pro-active at answering questions in relation with the audit and the code logic. They demonstrated a good understanding of the Solana ecosystem and of security hygiene.

Nonetheless, the audit demonstrated the presence of 2 important vulnerabilities. They were fixed at the end of the audit.

In the future, it should be considered to improve the code readability by factorizing some part of the code that are heavily duplicated. Such as the mechanisms for updating the rewards pools and the computation of the user rewards. In addition to complexifying the code, duplicate code is often the reason of the introduction of security vulnerabilities as developers sometimes forget to update all the duplicates entries when adding new features.