CoinFabrik

Smithii Mantis Protocol Audit

June 2024

By CoinFabrik

v202311

Executive Summary

CoinFabrik was asked to audit the contracts for the Mantis Protocol project.

Project Description

The protocol is intended to be a Solana on-chain platform to conduct token sales. Any user (called `authority`) is allowed to create a `Launch`, a time-dependent one-token offering (with a start and an end-date), during which any user is allowed to buy.

A `Launch` can be divided into two phases, namely whitelist phase and public phase (the whitelist phase is not mandatory). The `authority` defines fixed prices and selling conditions for each phase.

At `Launch` creation, the protocol transfers the maximum possible amount of tokens to be sold from the authority's token account to a vault controlled by the program. During the sale and until claim/withdrawal tokens are under custody of the program.

Only USDT, USDC, or SOL coins are permitted for issuing payments. The specific coin to be used is determined by the authority.

The protocol charges fees at two stages. Fees are sent to Smithii's controlled accounts:

- At `Launch` creation, a fixed amount of SOL: 200,000,000 lamports if the `Launch` has a whitelist phase, 100,000,000 if not. Paid by `authority`.
- At each purchase, a 2.5% percentage of the purchase amount. Paid by the buyer.

While fees and payment amounts are immediately transferred to Smithii / `authority` controlled accounts, bought tokens are locked until the `Launch` is over, when buyers are allowed to claim. There is no time limit for buyers to make their claim.

Notably, the amount of tokens corresponding to a purchase is not calculated during the `buy()` instruction but during the `claim()` process. The `buy()` instruction merely records and stores the purchase(s) amount(s) in a user's state PDA account.

`Launch` settings can be edited by its `authority` after creation, up to its first phase start date.

One day after the sale is over, the `authority` can withdraw any unsold remaining tokens from the program's vault.
Update commit `ee36cbb97f299e989437f4f8b65ab37cdef09085`: Now there is no more a one day timelock rule for withdrawing. The `authority` can withdraw remaining

tokens right after the `Launch` is over, either because `public_phase.end time` or the `hardcap` has been reached.

Summary

During this audit we found two critical issues, one high severity issue and one minor issue. Also, several enhancements were proposed.

Scope

The audited files are from the git repository located at
https://github.com/SmithiiDev/mantis_protocol

The audit was conducted (in order) on the following commits:

1. `B47015b2b38abda5c73621dd1e2ac784b7632ce`
2. `ee36cbb97f299e989437f4f8b65ab37cdef09085`

The scope for this audit includes and is limited to the following files:

- `src/instructions/buy.rs`: Buy instruction implementation
- `src/instructions/claim.rs`: Claim instruction implementation
- `src/instructions/edit.rs`: Edit instruction implementation
- `src/instructions/initialize.rs`: Initialize the launch
- `src/instructions/mod.rs`: Top file
- `src/instructions/withdraw.rs`: Withdraw tokens instruction
- `src/state/launch.rs`: Launch account
- `src/state/mod.rs`: Top file
- `src/state/user.rs`: User account
- `src/utils/mod.rs`: Top file
- `src/utils/transfers.rs`: Token transfer code
- `src/constants.rs`: Hardcoded values
- `src/errors.rs`: Error text
- `src/lib.rs`: Top file

No other files in this repository were audited. Its dependencies are assumed to work according to their documentation. Also, no tests were reviewed for this audit.

Methodology

CoinFabrik was provided with the source code and general documentation about the project. Our auditors spent two weeks auditing the source code provided, which includes understanding the context of use, analyzing the boundaries of the expected behavior of each contract and function, understanding the implementation by the development team (including dependencies beyond the scope to be audited) and identifying possible situations in which the code allows the caller to reach a state that exposes some vulnerability. Without being limited to them, the audit process included the following analyses.

- Arithmetic errors
- Race conditions
- Reentrancy attacks
- Misuse of block timestamps
- Denial of service attacks
- Excessive gas usage
- Missing or misused function qualifiers
- Needlessly complex code and contract interactions
- Poor or nonexistent error handling
- Insufficient validation of the input parameters
- Incorrect handling of cryptographic signatures
- Centralization and upgradeability

Findings

In the following table we summarize the security issues we found in this audit. The severity classification criteria and the status meaning are explained below. This table does not include the enhancements we suggest to implement, which are described in a specific section after the security issues.

ID	Title	Severity	Status
CR-01	Authority Can Withdraw All Tokens	Critical	Resolved
CR-02	Authority Can Freeze/Mint Sold Tokens	Critical	Resolved
HI-01	Validation Bypass on Launch Phase	High	Resolved
MI-01	Inconsistent Parameter Validation	Minor	Resolved

Severity Classification

Security risks are classified as follows:

- **Critical:** These are issues that we manage to exploit. They compromise the system seriously. Blocking bugs are also included in this category. They must be fixed **immediately**.
- **High:** These refer to a vulnerability that, if exploited, could have a substantial impact, but requires a more extensive setup or effort compared to critical issues. These pose a significant risk and **demand immediate attention**.
- **Medium:** These are potentially exploitable issues. Even though we did not manage to exploit them or their impact is not clear, they might represent a security risk in the near future. We suggest fixing them **as soon as possible**.
- **Minor:** These issues represent problems that are relatively small or difficult to take advantage of, but might be exploited in combination with other issues. These kinds of issues do not block deployments in production environments. They should be taken into account and be fixed **when possible**.

Issues Status

An issue detected by this audit has one of the following statuses:

- **Unresolved**: The issue has not been resolved.
- **Acknowledged**: The issue remains in the code, but is a result of an intentional decision. The reported risk is accepted by the development team.
- **Resolved**: Adjusted program implementation to eliminate the risk.
- **Partially resolved**: Adjusted program implementation to eliminate part of the risk. The other part remains in the code, but is a result of an intentional decision.
- **Mitigated**: Implemented actions to minimize the impact or likelihood of the risk.

Critical Severity Issues

CR-01 Authority Can Withdraw All Tokens

Location:
- `src/instructions/withdraw.rs:40`

Classification:
- CWE-675: Multiple Operations on Resource in Single-Operation Context[1]

The `withdraw()` function allows the `authority` to remove unsold tokens from the vault. The amount of tokens to withdraw is calculated as:

[1] https://cwe.mitre.org/data/definitions/675.html

```
amount_to_withdraw =  amount_to_sell - alloc_to_buyers
```

Note that in no part of the code, this calculation checks if all buyers have actually claimed the tokens. As a result, if `amount_to_withdraw` is less or equal than `alloc_to_buyers`, and if users have not claimed yet, calling `withdraw()` multiple times will send allocated tokens to the `authority` account, producing a not enough funds error later when users execute `claim()`. Under certain scenarios, this could lead to a complete removal of vault's funds.

Recommendation

Allowing only one call to `withdraw()` will avoid this issue.

Alternatively, improve the `amount_to_withdraw` calculation with code similar to this:

```
assert!(VAULT_BALANCE > alloc_to_buyers, "Nothing to withdraw");
amount_to_withdraw = VAULT_BALANCE - alloc_to_buyers;
```

Status

Resolved. Fixed according to the recommendation by **allowing only one call to withdraw**. Also, the Smithii team spotted a potential problem with our alternative solution. Checked on commit ee36cbb97f299e989437f4f8b65ab37cdef09085.

CR-02 Authority Can Freeze/Mint Sold Tokens

Location:

- `src/state/launch.rs:40`

Classification:

- CWE-413: Improper Resource Locking[2]

Since there are no checks during `Launch` creation to enforce that Mint's mint/freeze authorities have been revoked, buyers are exposed to:

- Token accounts frozen upon claim.
- Token´s total supply manipulation.

Also notice that the authority of the `Launch` not necessarily is the issuer of the token used for the sale.
Additionally, the total supply of the token is not checked. This exposes the protocol to pump and dump schemes.

[2]https://cwe.mitre.org/data/definitions/413.html

Recommendation

Add a check to prevent mint/freeze of tokens on launch initialization.

Status

Resolved. Fixed by requiring that a Launch's token mint and freeze authorities be revoked upon initialization. However, it is worth mentioning that the amount of the total supply locked in the Launch's vault remains unchecked.

Checked on commit ee36cbb97f299e989437f4f8b65ab37cdef09085.

High Severity Issues

HI-01 Validation Bypass On Launch Phase

Location:

- `src/state/launch.rs:60,150,162`

Classification:

- CWE-20: Improper Input Validation[3]

The `init()` function at `launch.rs` perform several validations on the incoming parameters:

```
    pub fn init(
        &mut self,
        authority: Pubkey,
        mint: Pubkey,
        hardcap: u64,
        softcap: u64,
        whitelist_phase: Phase,
        whitelist_limit: u64,
        public_phase: Phase,
        payment_method: PaymentMethod,
    ) -> Result<()> {
...
        if self.whitelist_phase.end_date > self.public_phase.start_date {
            return Err(error!(LaunchError::WhitelistEndDateInitializationError));
        }
```

But we can see that the validations are made over `self.whitelist_phase` instead of the incoming parameter `whitelist_phase`. Same incorrect validation is made on the `edit_pool()` function at lines 150 and 162 of the same file. Not only are validations bypassed but a problematic effect of this issue is that the white phase parameters cannot be edited if the original price of the whitelist phase is zero.

[3]https://cwe.mitre.org/data/definitions/20.html

Recommendation

Correctly validate the `whitelist_phase` variable in all functions.

Status

Resolved. Both **`init()` and `edit_pool()` now properly use the incoming values**. It is worth noting that for `edit_pool()`, it is allowed to edit the `white_phase` params only if it was created during initialization. Therefore, the fix correctly checks if `self.whitelist_price > 0` (used as a flag for `whitelist_phase` existence) and then uses the incoming values to edit the `whitelist_phase`. This is now well documented in the source code by the Smithii team.

Checked on commit ee36cbb97f299e989437f4f8b65ab37cdef09085

Medium Severity Issues

No issues found.

Minor Severity Issues

MI-01 Inconsistent Parameter Validation

Location:
- `src/instructions/launch.rs:57,147`

Classification:
- CWE-20: Improper Input Validation[4]

The `init()` function validate the hardcap in this way:

```
        if hardcap <= softcap { ...}
```

While the `edit_pool()` function validates it differently:

```
        if hardcap < softcap { ...}
```

As a consequence, some values that can be initialized, cannot be set at later stages. This issue criticality is lowered to minor because the `softcap` value is not used in this version of the protocol.

[4] https://cwe.mitre.org/data/definitions/20.html

Recommendation

Perform validations in a separate function so they are consistent across calls.

Status

Resolved. The **<= was added to edit_pool()**. Checked on commit ee36cbb97f299e989437f4f8b65ab37cdef09085.

Enhancements

These items do not represent a security risk. They are best practices that we suggest implementing.

ID	Title	Status
EN-01	Consolidate Constants	Implemented
EN-02	Remove Unused Accounts	Implemented

EN-01 Consolidate Constants

Location:

- `src/constants.rs`

While many constants are located at `constants.rs,` other constants like Launch creation fees are hardcoded in the body of the `initialize()` instruction.

Recommendation

Consolidate all constants in a single file.

Status

Implemented. Checked on commit ee36cbb97f299e989437f4f8b65ab37cdef09085.

EN-02 Remove Unused Accounts

Location:

- `src/instructions/withdraw.rs:35`
- `src/instructions/edit.rs:32`

The `AssociatedToken` and `SystemProgram` accounts are not necessary in the context of the `withdraw()` and `edit()` instructions.

Recommendation

Remove all unused accounts.

Status

Implemented. Checked on commit ee36cbb97f299e989437f4f8b65ab37cdef09085.

Other Considerations

The considerations stated in this section are not right or wrong. We do not suggest any action to fix them. But we consider that they may be of interest to other stakeholders of the project, including users of the audited contracts, token holders or project investors.

Centralization

No centralized designs or privileges were found on the contracts.

Upgrades

All Solana programs are upgradable by its upgrade-authority. By default, it is the address that deployed the program. The upgrade-authority is the only one entitled to change the upgrade-authority of a program.

A program can be made immutable by setting its upgrade-authority to None. Once nullified, it cannot be changed again.

This can be done during deployment or after, at any moment.

Therefore, this program should be considered upgradable until deployment and manually checking the upgrade-authority status.

Privileged Roles

These are the privileged roles that we identified on each of the audited contracts.

Authority

The authority role is granted to the address that initializes a Launch, and it applies to that specific Launch. Upon initialization, the authority sets the Launch and its Phases parameters, including Mint, price, hard cap, soft cap, payment coin, start date, end date, etc.

The authority pays Launch's creation fees, funds Launch's token vault and receives payments on each purchase, after deducting protocol fees.

Only the authority can edit the Launch settings, up to the start date of the first phase. Once the Launch has started, the edit instruction cannot longer be called.

Only the authority can withdraw unsold remaining tokens from the program's vault, once the Launch has ended.

Note that the authority does not necessarily have to be the token creator or its mint/freeze authority, and the percentage of the total supply of the token that is locked in the Launch's vault is not checked.

User

The user role is assigned to any address that makes a token purchase during a Launch. It is only relevant for that particular Launch, but the same address can be a user on different Launches.

There is no limit to the amount of purchases a user can make. Purchases can only be made while the Launch is live.

After Launch's public phase end date or after its hard cap has been reached, a user can claim.

Each user can execute the claim instruction only once.

Additional Comments

- Project is not compatible with Token-2022
- The whitelist phase is only a period of time, there is no real list of users, it's as public as the public phase.
- The program is not compatible with Legacy Token Accounts
- There is extensive use of msg!(...) which is costly.

Update July 2nd, 2024: the number of msg!(...) has been significantly reduced. Checked on commit ee36cbb97f299e989437f4f8b65ab37cdef09085

Changelog

- 2024-06-18 – Initial report based on commit `b47015b2b38abda5c73621dd1e2ac784b7632ce1`
- 2024-07-02 - Fixes review on commit ee36cbb97f299e989437f4f8b65ab37cdef09085

Disclaimer: This audit report is not a security warranty, investment advice, or an approval of the Smithii Mantis Protocol project since CoinFabrik has not

reviewed its platform. Moreover, it does not provide a smart contract code faultlessness guarantee.