Avalaunch Audit <>Allocation Staking and Sales

September 2, 2021 Heloisa Ceni No Comments

Reading Time: 4 minutes

0
Shares

Contents

Introduction

CoinFabrik was asked to audit the contracts for the Avalaunch project. First we will provide a summary of our discoveries and then we will show the details of our findings.

Summary

The contracts audited are from the Github repository at https://github.com/avalaunch-app/xava-protocol/. The audit is based on the commit fd252f8b9b0283d245d9d561130fe789ff08dfe9. Next, developers fixed issues and we re-checked commit ac00d9c0d66b4abc3a892530c9dfc349811050ad.

Contracts

The audited contracts are:

- contracts/AllocationStaking.sol
- contracts/sales/SalesFactory.sol
- contracts/sales/AvalaunchSale.sol

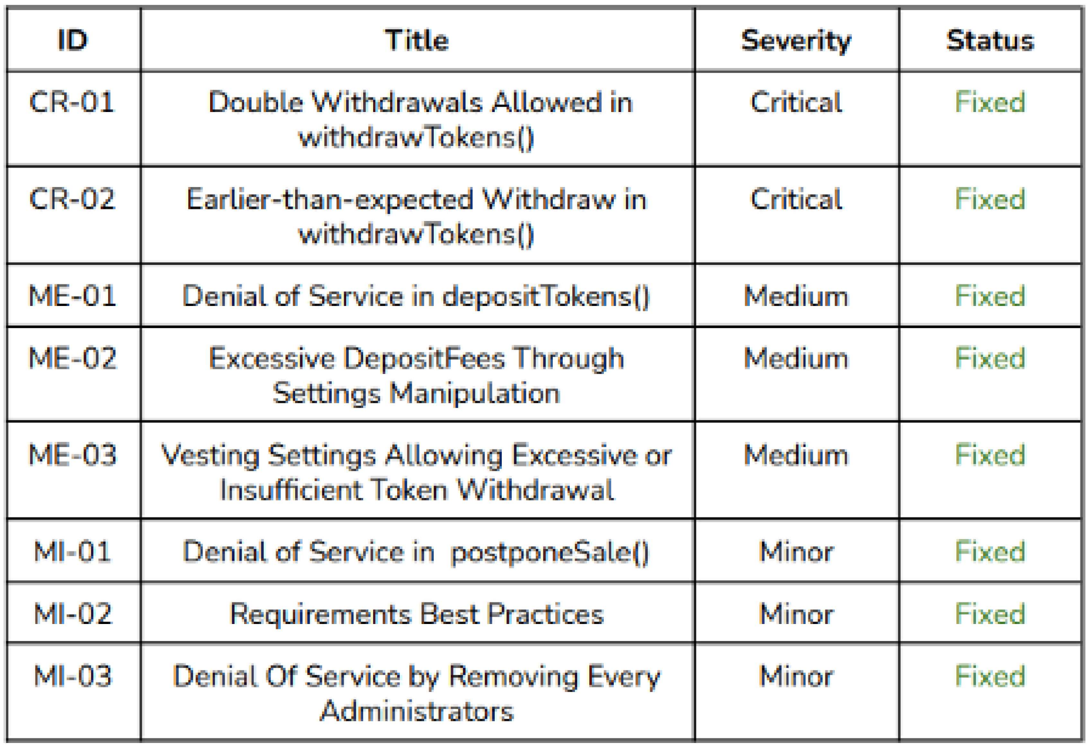
Analyses

The following analyses were performed:

- Misuse of the different call methods
- Integer overflow errors
- Division by zero errors
- Outdated version of Solidity compiler
- Front running attacks
- Reentrancy attacks
- Misuse of block timestamps
- Softlock denial of service attacks
- Functions with excessive gas cost

● Missing or misused function qualifiers
● Needlessly complex code and contract interactions
● Poor or nonexistent error handling
● Failure to use a withdrawal pattern
● Insufficient validation of the input parameters
● Incorrect handling of cryptographic signatures

Findings and Fixes

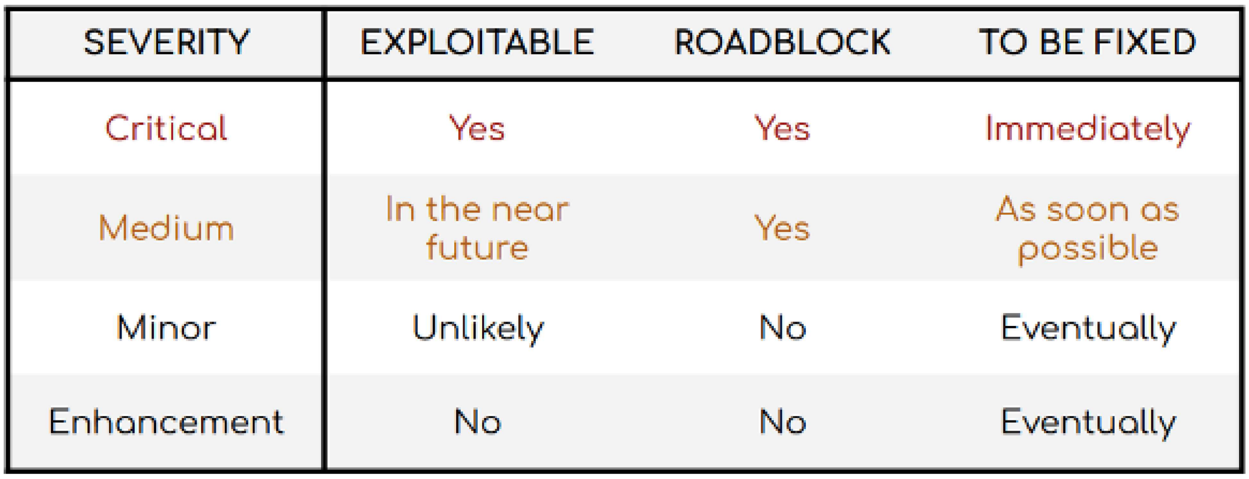


ID	Title	Severity	Status
CR-01	Double Withdrawals Allowed in withdrawTokens()	Critical	Fixed
CR-02	Earlier-than-expected Withdraw in withdrawTokens()	Critical	Fixed
ME-01	Denial of Service in depositTokens()	Medium	Fixed
ME-02	Excessive DepositFees Through Settings Manipulation	Medium	Fixed
ME-03	Vesting Settings Allowing Excessive or Insufficient Token Withdrawal	Medium	Fixed
MI-01	Denial of Service in postponeSale()	Minor	Fixed
MI-02	Requirements Best Practices	Minor	Fixed
MI-03	Denial Of Service by Removing Every Administrators	Minor	Fixed

Severity Classification

Security risks are classified as follows:
● **Critical**: These are issues that we manage to exploit. They compromise the system seriously. They must be fixed immediately.
● **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 can 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.
● **Enhancement:** These kinds of findings do not represent a security risk. They are

best practices that we suggest to implement.
This classification is summarized in the following table:



SEVERITY	EXPLOITABLE	ROADBLOCK	TO BE FIXED
Critical	Yes	Yes	Immediately
Medium	In the near future	Yes	As soon as possible
Minor	Unlikely	No	Eventually
Enhancement	No	No	Eventually

Issues Found by Severity

Critical severity

CR-01 Double Withdrawals Allowed in withdrawTokens()

The function fails to mark a portion as withdrawn. This may be due to a typo in the line

```
p.isPortionWithdrawn&#91;portionId];
```

As a result, any user can withdraw the same portion more than once.

Recommendation

Set

```
p.isPortionWithdrawn&#91;portionId] = true;
```

Solution

The issue was fixed following the recommendation.

CR-02 Earlier-than-expected Withdraw in withdrawTokens()

When withdrawing tokens for a given portionID the function checks that

```
vestingPortionsUnlockTime&#91;portionId] >= block.timestamp
```

so that all portions can be vested at the start.

Recommendation

Check the opposite condition.

Solution

The issue was fixed following the recommendation.

Medium severity

ME-01 Denial of Service in depositTokens()

The function depositTokens() in AvalaunchSale requires that

```
sale.token.balanceOf(address(this)) == 0
```

Hence, a malicious user could transfer a minimal amount of tokens to the contract so the above condition is not met.

Recommendation

Instead, use

```
require(!sale.tokensDeposited, message);
```

Solution

The issue was fixed following the recommendation

ME-02 Excessive DepositFees Through Settings Manipulation

Both when the AllocationStaking contract is initialized and when `setDepositFee()` is called, `_depositFeePercent` may be set to any value independently of `_depositFeePrecision`. The same consideration should apply during initialization (`_depositFeePercent` must be smaller than the public variable `depositFeePrecision = 10e8`).

Solution

The issue was fixed following the recommendation.

ME-03 Vesting Settings Allowing Excessive or Insufficient Token Withdrawal

In `setVestingParams()` no checks are made that the sum of _percents is 100. For example, `if _percents[0] = 200`, when a user calls `withdrawTokens()` he would receive twice the amount bought:

```
p.amountBought.mul(vestingPercentPerPortion&#91;portionId]).div(10 0)
```

Similarly, if the sum is smaller than 100, the user would be prevented from vesting all the tokens he bought.

Recommendation

Require that the sum of the percentages is 100.

Solution

The issue was fixed following the recommendation.

Minor Severity

MI-01 Denial of Service in postponeSale()

Using a large value for `timeToShift` in `postponeSale()` may shift `round.startTime` to be bigger than `sale.saleEnd` rendering the sale useless.

Recommendation

Require that `round.startTime + timeToShift < sale.saleEnd.`

Solution

The issue was fixed following the recommendation.

MI-02 Requirements Best Practices

In different places in the contacts we see requirements of the following sort:

```
require(condition == true, msg)
require(condition == false, msg)
```

which although allowed, are not preferred. Instead use require(condition, msg) and require(!condition, msg) respectively. These may use less gas.

Solution

Fixed.

MI-03 Denial Of Service by Removing All the Administrators

The function `removeAdmin()` can be called arbitrarily and could be used to remove every administrator.

Recommendation

Make sure that this does not happen by requiring that `len(Admin) > 1`.

Solution

Fixed. The recommended change was applied.

Enhancements

Constants and Code Legibility

Use one in

```
uint256 amountOfTokensBuying =
(msg.value).mul(10**18).div(sale.tokenPriceInAVAX);
```

instead of 10**18. Similarly, the value 1e36 is used throughout the AllocationStake contract. Consider replacing this with a constant that can be called by its name.

Small Precision In Vesting Scheme

The precision for the percentage points is set to 1 (so that all values go from 1 to 100). Using a larger number could accommodate more flexibility.

Gas Optimization in the Round Structure

The struct Round defined in AvalaunchSale.sol has two unspecified uints (uint startTime; uint maxParticipation;) and in the struct Registration all objects can fit in 256.

Unnecessary Code in setRounds()

The function makes use of a for loop which verifies conditions for every round. However, in some cases, only the first and last rounds need to be checked. Specifically,

```
require(startTimes&#91;0] > registration.registrationTimeEnds);
require(startTimes&#91;0] >= block.timestamp);

```

since the code also requires that `startTimes[i] > startTimes[i-1]` for every i. Analogously, checking

```
require(startTimes&#91;last] &lt; sale.saleEnd);
```

is sufficient.

Gas Optimization in `massUpdatePool()` and `compound()`

Consider factoring updatePoolWithFee() so that the computation of nrOfSeconds is only done once (and not once per pool.) This same behaviour is seen in compound() which makes two calls to updatePoolWithFee() (the first one via updatePool()). In the second, nrOfSeconds will be 0 and most of the logic is useless. Consider
decoupling the two functionalities in order to save gas

Gas Optimization in PoolInfo

A gas optimization for PoolInfo could be done by reducing the size of the lastRewardTimestamp and allocPoint. For example, fitting each into an uint128 (so the two fit into 32 bytes).
Note that, since 2^32 – 1 is a timestamp for February 2106, 32 bits are enough for a timestamp and 128 is obviously sufficient. If less than 2^128 allocation points are sufficient per pool, then restructuring PoolInfo as we proposed can be done.

Conclusion

We found the contracts to be simple and straightforward. Documentation is scarce. Two critical vulnerabilities were found which could allow a user to withdraw tokens in excess or before their time. These issues were fixed. Three medium severity issues were found, one of which allows a malicious user to make a sale contract unusable, and the two other which can be exercised when the administrator sets incorrect parameters–so they are unlikely to be exploitable. All issues have been resolved.

MORE DETAILS

Disclaimer: This audit report is not a security warranty, investment advice, or an approval of the Avalaunch project since CoinFabrik has not reviewed its platform. Moreover, it does not provide a smart contract code faultlessness guarantee.

Related Posts

Katana Smart Contract Audit

CoinFabrik was asked to audit the contracts for the Katana project. First we will provide...

Stasis Token Smart Contract Audit

STASIS

Coinfabrik has been hired to audit the smart contracts for Stasis sale, the Stable Euro…

BLOX STAKING Audit: Vesting and DEX Contracts

Introduction CoinFabrik was asked to audit the contracts for the Vesting for Blox Staking. First…

YFFII Smart Contract Audit

CoinFabrik was asked to audit the contracts for the YFFII project. First we will provide…

Categorized in: Smart Contract Audit, Smart Contracts

ALSO ON **COINFABRIK**

BLOX STAKING Audit: Vesting and DEX …

a month ago

Introduction CoinFabrik was asked to audit the contracts for the Vesting for Blox …

iCherry Smart Contract Audit

a year ago • 1 comment

CoinFabrik was asked to audit the contracts for the ICherry Finance project. …

Known Origin Digital Asset Audit

a month ago • 1 comment

Introduction CoinFabrik was asked to audit the contracts for the Known Origin …

An to

3 ye

The ine hav