Security Audit Report for AuroraStNear

Date: Mar 20th, 2022

Version: 1.0

Contact: contact@blocksecteam.com

BlockSec Audit Team

Digitally signed by BlockSec Audit Team
Date: 2022.03.20 17:54:44 +08'00'

Contents

Report Manifest

Item	Description
Client	Meta Pool
Target	AuroraStNear

Version History

Version	Date	Description
1.0	Mar 20th, 2022	First Release

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The audit scope only includes the AuroraStNear contract [1].

The auditing process is iterative. Specifically, we will further audit the commits that fix the founding issues. If there are new issues, we will continue this process. Thus, there are multiple commit SHA values referred in this report. The commit SHA values before and after the audit are shown in the following.

Before and during the audit

Contract Name	Stage	Commit SHA
AuroraStNear	Initial	`bd4a4c5b1baab4139d3b51b33248b822bb1fcb06`

After

Project	Commit SHA
AuroraStNear	`0aa4b634176d7fea3997a3d3252fffe8791e829a`

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

[1] https://github.com/Narwallets/aurora-swap/blob/master/contracts/AuroraStNear.sol

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Software Security

- Reentrancy
- DoS
- Access control
- Data handling and data flow
- Exception handling
- Untrusted external call and control flow
- Initialization consistency
- Events operation
- Error-prone randomness
- Improper use of the proxy system

1.3.2 DeFi Security

- Semantic consistency
- Functionality consistency
- Access control
- Business logic
- Token operation
- Emergency mechanism
- Oracle security
- Whitelist and blacklist
- Economic impact
- Batch transfer

1.3.3 NFT Security

- Duplicated item
- Verification of the token receiver
- Off-chain metadata security

1.3.4 Additional Recommendation

- Gas optimization
- Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. Accordingly, the severity measured in this report are classified into four categories: **High**, **Medium**, **Low** and **Undetermined**.

Furthermore, the status of a discovered issue will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The issue has been received by the client, but not confirmed yet.
- **Confirmed** The issue has been recognized by the client, but not fixed yet.
- **Fixed** The issue has been confirmed and fixed by the client.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

Chapter 2 Findings

In total, we find 2 potential issues in the smart contract. We also have 3 recommendations, as follows:

- High Risk: 0
- Medium Risk: 0
- Low Risk: 2
- Recommendations: 3

The details are provided in the following sections.

ID	Severity	Description	Category	Status
1	Low	*Mismatched Event and Function*	Software Security	Fixed
2	Low	*Missing check on* `_stNearPrice`	DeFi Security	Fixed
3	-	*Redundant Code*	Recommendation	Fixed
4	-	*Potential Centralization Problems*	Recommendation	Acknowledged
5	-	*A Stable and Efficient Oracle is Required*	Recommendation	Confirmed

2.1 Software Security

2.1.1 Mismatched Event and Function

Status Fixed.

Description This issue is introduced in or before the initial commit.

```
function withdrawstNEAR(uint256 _amount)
  external
  onlyRole(DEFAULT_ADMIN_ROLE)
  nonReentrant
{
  require(
    stNear.balanceOf(address(this)) >= _amount,
    "Not enough stNEAR in buffer"
  );
  if (_amount >= stNearAccumulatedFees) {
    stNearAccumulatedFees = 0;
  } else {
    stNearAccumulatedFees -= _amount;
  }
  stNear.safeTransfer(msg.sender, _amount);
  emit wNearWithdraw(msg.sender, _amount);
}
```

Listing 2.1: AuroraStNear.sol

Instead of emitting event `wNearWithdraw` in this function `withdrawstNEAR`, event `stNearWithdraw` should be used.

```
  event wNearWithdraw(address indexed admin, uint256 amount);
  event stNearWithdraw(address indexed admin, uint256 amount);
```

Listing 2.2: AuroraStNear.sol

Impact The contract will emit incorrect events.

Suggestion I Emit the correct event `stNearWithdraw` in function `withdrawstNEAR`.

2.2 DeFi Security

2.2.1 Missing check on _stNearPrice

Status Fixed.

Description This issue is introduced in or before the initial commit. Missing price check in `constructor` and `setstNEARPrice()`. The `_stNearPrice` may be set to 0 or some other unreasonable values.

```
constructor(
  IERC20Metadata _wNear,
  IERC20Metadata _stNear,
  uint256 _stNearPrice
) {
  wNear = _wNear;
```

```
    stNear = _stNear;
    stNearPrice = _stNearPrice;
    wNearSwapFee = 30;
    stNearSwapFee = 10;
    _setupRole(DEFAULT_ADMIN_ROLE, msg.sender);
    grantRole(OPERATOR_ROLE, msg.sender);
  }
```

Listing 2.3: AuroraStNear.sol

```
function setstNEARPrice(uint256 _stNearPrice)
  external
  onlyRole(OPERATOR_ROLE)
{
  stNearPrice = _stNearPrice;
}
```

Listing 2.4: AuroraStNear.sol

Impact The whole swapping logic may be wrong if `_stNEARPrice` is unchecked.

Suggestion I Add a reasonable threshold to check the value of `_stNearPrice`.

2.3 Additional Recommendation

2.3.1 Redundant Code

Status Fixed.

Description This issue is introduced in or before the initial commit. These two events are not used in this contract.

```
    event wNearDeposit(address indexed admin, uint256 amount);
    event stNearDeposit(address indexed admin, uint256 amount);
```

Listing 2.5: AuroraStNear.sol

Suggestion I Remove the unused codes.

2.3.2 Potential Centralization Problems

Status Acknowledged.

Description This issue is introduced in or before the initial commit. This project has potential centralization problems. Functions like `withdrawwNEAR` and `withdrawstNEAR` allow the admin of the contract to withdraw any amount of wNEAR/stNEAR.

Suggestion I It is recommended to introduce a decentralization design in the contract, such as multi-signature or DAO. The project owner needs to ensure the private key of the ADMIN_ROLE/OPERATOR_ROLE should not be leaked.

Feedback from the project The contract works as a buffer to allow an "easy swap, NEAR->stNEAR" for Aurora users avoiding the long road of bridging back and forth to NEAR in order to stake in Metapool. All the wNEAR and stNEAR in the contract belongs to the Admin. The user deposits wNEAR and receives stNEAR or vice versa, but the contract at no point holds users funds. The "buffer" of wNEAR and stNEAR must be filled by the admin that is a bot (conveyor-belt-bot), so it is necessary for the bot to be able to withdraw wNEAR, bridge to NEAR, stake in metapool, bridge the stNEAR back to Aurora and deposit the stNEAR in the aurora contract. Since the bot is automated and performs this action every 5 minutes, it is not suitable to have a multisig or a DAO schema for the Admin signature.

2.3.3 A Stable and Efficient Oracle is Required

Status Confirmed.

Description This issue is introduced in or before the initial commit. The contract highly depends on a stable and efficient Oracle. The oracle should poke the `stNearPrice` timely and precisely.

```
function setstNEARPrice(uint256 _stNearPrice)
    external
    onlyRole(OPERATOR_ROLE)
{
    stNearPrice = _stNearPrice;
}
```

Listing 2.6: AuroraStNear.sol

Feedback from the project There is a stable and efficient Oracle under our control that updates the price on every stNEAR price change on the source: Meta Pool NEAR native. stNEAR is a NEAR derivative and has a price determined by the amount of staked NEAR plus rewards in Meta Pool (stNEAR_price = NEAR_Staked/stNEAR_Minted), so there's a precise price, that increases at the end of every epoch when staking rewards are contabilized. As soon as the stNEAR price is updated in the NEAR-Native Meta Pool contract by the Metapool bot, the price is also updated in this Aurora contract, keeping both values in sync.