Noodles Security Review

Pashov Audit Group

Conducted by: Hals, Kurosaki, shaflow, 0xTheBlackPanther, dobrevaleri, Silvermist

March 11th 2025 - March 14th 2025

Contents

1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Introduction

A time-boxed security review of the **noodles-fun/contracts** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Noodles

Noodles.Fun turns Twitter (X) accounts into tradeable tokens using bonding curves, enabling users to buy, sell, and use these tokens for promotions like shoutouts or pinned tweets, fostering deeper engagement with Key Opinion Leaders (KOLs). Each account’s token operates on its own bonding curve, allowing instant trading without liquidity pools or order books, while creators can earn ETH and boost token value by buying back and burning their tokens.

5. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

review commit hash - f276d0ddf8a21ce02fa8641a2e46ff8fc3c6db74

fixes review commit hash - 980ad2f02c22599e966a940c9c47ff04e03af6f1

Scope

The following smart contracts were in scope of the audit:

- `VisibilityCredits`
- `VisibilityServices`

7. Executive Summary

Over the course of the security review, Hals, Kurosaki, shaflow, 0xTheBlackPanther, dobrevaleri, Silvermist engaged with Noodles to review Noodles. In this period of time a total of **5** issues were uncovered.

Protocol Summary

Protocol Name	Noodles
Repository	https://github.com/noodles-fun/contracts
Date	March 11th 2025 - March 14th 2025
Protocol Type	Social Account Tokenization

Findings Count

Severity	**Amount**
Medium	2
Low	3
Total Findings	**5**

Summary of Findings

ID	Title	Severity	Status
[M-01]	Incorrect bonus percentages for partner and referrer	Medium	Resolved
[M-02]	Missing slippage protection on sellCredits()	Medium	Resolved
[L-01]	Missing restriction on request data size during request lifecycle	Low	Resolved
[L-02]	Insufficient validation allows pre-filling data for invalid execution nonces	Low	Resolved
[L-03]	Insufficient validation allows ETH transfers on visibility credits payment	Low	Resolved

8. Findings

8.1. Medium Findings

[M-01] Incorrect bonus percentages for partner and referrer

Severity

Impact: Low

Likelihood: High

Description

In the `VisibilityCredits` contract, the `PARTNER_FEE` and `PARTNER_REFERRER_BONUS` are incorrectly set to 250 ppm, which represents 0.025%. However, as per the comment in the contract, these values should be set to 0.25%. This misconfiguration will result in the partner and referrer receiving a less bonus from the total cost of the bought credits than intended.

```
uint16 public constant PARTNER_FEE = 250;
// 0.25% bonus for the partner/marketing agency
// if linked to a referrer (deduced from protocol fee)
uint16 public constant PARTNER_REFERRER_BONUS = 250;
// 0.25% bonus for the referrer
// if linked to a partner (deduced from protocol fee)
```

Recommendations

Update the values of `PARTNER_FEE` and `PARTNER_REFERRER_BONUS` to 2500 ppm (0.25%) to match the intended values.

[M-02] Missing slippage protection on sellCredits()

Severity

Impact: Medium

Likelihood: Medium

Description

Users trading credit tokens are prone to slippage by bots. Since VisibilityCredits.sol utilises a bonding curve, the price of credit tokens increase and decrease as the total supply increases and decreases.

Slippage on sell:

- Alice submits sellCredits() transaction.
- The transaction remains the mempool for sometime.
- During this period, another sellCredits() transaction executes (unintentionally by other users or intentionally by bots), thus decreasing the trading cost.
- Alice's transaction goes through but receives lesser native tokens than intended.

Recommendations

Provide the users with the option to pass in a `minAmountOut` parameter on sells. Check the trading cost against this parameter to ensure the user receives what they intended to.

8.2. Low Findings

[L-01] Missing restriction on request data size during request lifecycle

The functions `requestServiceExecution`, `addInformationForServiceExecution`, `acceptServiceExecution`, `cancelServiceExecution`, and `disputeServiceExecution` allow users to input string data of arbitrary length.

```
function requestServiceExecution(
        uint256 serviceNonce,
        string calldata requestData
    ) external payable {

    function addInformationForServiceExecution(
        uint256 serviceNonce,
        uint256 executionNonce,
        string calldata informationData
    ) external {

    function acceptServiceExecution(
        uint256 serviceNonce,
        uint256 executionNonce,
        string calldata responseData
    ) external {

    function cancelServiceExecution(
        uint256 serviceNonce,
        uint256 executionNonce,
        string calldata cancelData
    ) external {

    function disputeServiceExecution(
        uint256 serviceNonce,
        uint256 executionNonce,
        string calldata disputeData
    ) {
```

The lack of a maximum data size restriction could lead to malicious actors sending requests with large amounts of invalid data. These data would ultimately appear on the frontend, resulting in a poor user experience.

[L-02] Insufficient validation allows pre-filling data for invalid execution nonces

As per the documentation, function addInformationForServiceExecution() is mostly intended to be used by the creator to upload the information data about the X post created for a specific proposal (i.e. executionNonce). The issue is here is that the function does not validate the executionNonce to be less than the global `executionNonces` value. This allows the creator or dispute resolver to pre-populate information data for future nonces that are not existent yet. This does not seem to pose an on-chain threat currently, but the function does emit the `ServiceExecutionInformation`, which could be consumed differently by off-chain processes. Such behaviour should be disallowed, as it is not part of the intended flow.

```
function addInformationForServiceExecution(
        uint256 serviceNonce,
        uint256 executionNonce,
        string calldata informationData
    ) external {
        VisibilityServicesStorage storage $ = _getVisibilityServicesStorage();

        Service storage service = $.services[serviceNonce];
        Execution storage execution = service.executions[executionNonce];
```

[L-03] Insufficient validation allows ETH transfers on visibility credits payment

In the payable function requestServiceExecution(), when the payment type is the credit token, we do not ensure that the msg.value is 0. Due to this, if ETH is sent along with the call, it would remain locked in the contract. The sanity check `if (msg.value != 0) revert();` should be added to prevent any user mistakes from occurring during service requests.

```
if (paymentType == PaymentType.VISIBILITY_CREDITS) {
            /// @dev it reverts if not enough credits
            $.visibilityCredits.transferCredits(
                service.visibilityId,
                msg.sender,
                address(this),
                service.creditsCostAmount
            );
```