BLOCKSEC

Security Audit
Report for Noah V3

Date: January 7, 2025 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	NoaharkEVM
Target	Noah V3

Version History

Version	Date	Description
1.0	January 7, 2025	First release

Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2025-01-07 07:25:59+0000

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

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

The focus of this audit is on the Noah V3 of the NoaharkEVM [1]. The Noah V3 protocol is an open source decentralized exchange that allows users to trade freely on the chain. It is a DeFi infrastructure open to everyone. As with most AMMs, trades between specific asset pairs are facilitated by holding reserves of both assets. It sets the transaction price between them based on the size of its reserves, and the price changes with the number of tokens in the pool. Any user is free to join or withdraw from the token liquidity pool, and becoming a liquidity provider or LP will bear part of the transaction risk in exchange for returns.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report.

Project	Version	Commit Hash
Noah V3	`Version 1`	`ccfce2fb659beed8ee039603525dad862383831a`
	`Version 2`	`c9b25292090a23488da171c5cf9aaa786bf0d854`

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

[1] https://github.com/NoaharkEVM/noah-v3

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.

1.3 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
* Permission management
* 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 and Common Weakness Enumeration. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

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

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

Chapter 2 Findings

In total, we found **one** potential security issue. Besides, we have **three** recommendations and **two** notes.

- Medium Risk: 1
- Recommendation: 3
- Note: 2

ID	Severity	Description	Category	Status
1	Medium	Lack of permission check in function `setFactory()`	DeFi Security	Confirmed
2	-	Allow the change of the `_baseTokenURI`	Recommendation	Fixed
3	-	Lack of invoking function `_disableInitializers()`	Recommendation	Fixed
4	-	Lack of check for pool in function `createIncentive()`	Recommendation	Fixed
5	-	Potential centralization risk	Note	-
6	-	`POOL_INIT_CODE_HASH` must be correct.	Note	-

The details are provided in the following sections.

2.1 DeFi Security

2.1.1 Lack of permission check in function `setFactory()`

Severity Medium

Status Confirmed

Introduced by `Version 1`

Description The function `setFactory()` sets the factory address for the contract `NoahV3PoolDeployer`, which in turn allows for the deployment of pools via the function `deploy()`. However, this function lacks proper permission checks. Consequently, an attacker can exploit this vulnerability by invoking the function `setFactory()` immediately after the deployment of the contract `NoahV3PoolDeployer`, potentially compromising the control of the factory over the contract.

```
27  function setFactory(address _factory) external {
28      require(factory == address(0), "already initialized");
29
30      factory = _factory;
31
32      emit SetFactoryAddress(_factory);
33  }
```

Listing 2.1: projects/v3-core/contracts/NoahV3PoolDeployer.sol

Impact The contract `NoahV3Factory` may not be able to deploy new pools through the contract `NoahV3PoolDeployer` if the function `setFactory()` is invoked by attackers, causing the entire protocol to fail.

Suggestion Add a check to allow only the protocol owner to set the factory address.

Feedback from the Project After deployment and setting the factory address, a manual check will be performed to verify if the factory address is compliant.

2.2 Recommendations

2.2.1 Allow the change of the `_baseTokenURI`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `NonfungibleTokenPositionDescriptorOffChain`, once `_baseTokenURI` is set, it cannot be changed again.

```
11  contract NonfungibleTokenPositionDescriptorOffChain is INonfungibleTokenPositionDescriptor,
        Initializable {
12      using StringsUpgradeable for uint256;
13
14      string private _baseTokenURI;
15
16      function initialize(string calldata baseTokenURI) external initializer {
17          _baseTokenURI = baseTokenURI;
18      }
19
20      /// @inheritdoc INonfungibleTokenPositionDescriptor
21      function tokenURI(INonfungiblePositionManager positionManager, uint256 tokenId)
22          external
23          view
24          override
25          returns (string memory)
26      {
27          return bytes(_baseTokenURI).length > 0 ? string(abi.encodePacked(_baseTokenURI, tokenId
                .toString())) : "";
28      }
29  }
```

Listing 2.2: projects/v3-periphery/contracts/NonfungibleTokenPositionDescriptorOffChain.sol

Suggestion Implement the function `setBaseTokenURI()`.

2.2.2 Lack of invoking function `_disableInitializers()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contracts `NonfungibleTokenPositionDescriptorOffChain` and `NoahV3Staker`, the function `_disableInitializers()` is not invoked in the constructor. Invoking

this function prevents the contract itself from being initialized, thereby avoiding unexpected behaviors. It is a security best practice to prevent implementation contracts from being initialized independently.

```
21  contract NoahV3Staker is INoahV3Staker, Multicall, Initializable {
```

Listing 2.3: projects/v3-staker/contracts/NoahV3Staker.sol

```
11  contract NonfungibleTokenPositionDescriptorOffChain is INonfungibleTokenPositionDescriptor,
        Initializable {
```

Listing 2.4: projects/v3-periphery/contracts/NonfungibleTokenPositionDescriptorOffChain.sol

Suggestion Invoke the function `_disableInitializers()` in the constructor.

2.2.3 Lack of check for pool in function `createIncentive()`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The incentive key passed to the function `createIncentive()` could refer to a non-existent pool. If the user provides incorrect input, this may result in funds being stuck in the contract for a period of time.

```
97  function createIncentive(IncentiveKey memory key, uint256 reward) external override {
98      require(reward > 0, 'NoahV3Staker::createIncentive: reward must be positive');
99      require(
100         block.timestamp <= key.startTime,
101         'NoahV3Staker::createIncentive: start time must be now or in the future'
102     );
103     require(
104         key.startTime - block.timestamp <= maxIncentiveStartLeadTime,
105         'NoahV3Staker::createIncentive: start time too far into future'
106     );
107     require(key.startTime < key.endTime, 'NoahV3Staker::createIncentive: start time must be
            before end time');
108     require(
109         key.endTime - key.startTime <= maxIncentiveDuration,
110         'NoahV3Staker::createIncentive: incentive duration is too long'
111     );
112
113     bytes32 incentiveId = IncentiveId.compute(key);
114
115     incentives[incentiveId].totalRewardUnclaimed += reward;
116
117     TransferHelperExtended.safeTransferFrom(address(key.rewardToken), msg.sender, address(this),
            reward);
118
119     emit IncentiveCreated(key.rewardToken, key.pool, key.startTime, key.endTime, key.refundee,
            reward);
120 }
```

Listing 2.5: projects/v3-staker/contracts/NoahV3Staker.sol

Suggestion Add a check for the existence of the pool in the incentive key.

2.3 Notes

2.3.1 Potential centralization risk

Introduced by Version 1

Description The protocol owner can call the function `setStatus()` to set the status to disable swap, liquidity addition and liquidity removal separately for the entire protocol. Therefore, if the owner's private key is lost, tokens provided as liquidity in the pool may no longer be withdrawable.

```
114  function setStatus(
115      bool isSwapDisabled,
116      bool isAddLiquidityDisabled,
117      bool isDecreaseLiquidityDisabled
118  ) external override {
119      require(msg.sender == owner);
120      uint8 status = 0;
121      if (isSwapDisabled) status |= Status.SWAP_DISABLED;
122      if (isAddLiquidityDisabled) status |= Status.ADD_LIQUIDITY_DISABLED;
123      if (isDecreaseLiquidityDisabled) status |= Status.REMOVE_LIQUIDITY_DISABLED;
124      _status = status;
125      emit StatusChanged(isSwapDisabled, isAddLiquidityDisabled, isDecreaseLiquidityDisabled);
126  }
```

Listing 2.6: projects/v3-core/contracts/NoahV3Factory.sol

2.3.2 `POOL_INIT_CODE_HASH` must be correct.

Introduced by Version 1

Description The `POOL_INIT_CODE_HASH` must be correct, otherwise the computed address will be incorrect, causing the protocol to malfunction.

```
6  bytes32 internal constant POOL_INIT_CODE_HASH = 0
       x2c97634d58f0736d7103fbb4bc395e9df847283bcf281bdfe46be5001c43cf46;
```

Listing 2.7: projects/v3-periphery/contracts/libraries/PoolAddress.sol


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS