SMART CONTRACT AUDIT REPORT

for

Stader StakeManager

Prepared By: Xiaomi Huang

PeckShield

July 24, 2022

Document Properties

Client	Stader
Title	Smart Contract Audit Report
Target	Stader StakeManager
Version	1.0
Author	Luck Hu
Auditors	Luck Hu, Xuxian Jiang
Reviewed by	Xiaomi Huang
Approved by	Xuxian Jiang
Classification	Public

Version Info

Version	Date	Author(s)	Description
1.0	July 24, 2022	Luck Hu	Final Release
1.0-rc	July 18, 2022	Luck Hu	Release Candidate

Contact

For more information about this document and its contents, please contact PeckShield Inc.

Name	Xiaomi Huang
Phone	+86 183 5897 7782
Email	contact@peckshield.com

Contents

1 Introduction

Given the opportunity to review the design document and related smart contract source code of the StakeManager contract in the Stader protocol, we outline in the report our systematic approach to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistencies between smart contract code and design document, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts can be further improved due to the presence of several issues related to either security or performance. This document outlines our audit results.

1.1 About Stader

Stader specializes in non-custodial staking solutions for retail, enterprise customers, funds and blockchains. Currently, Stader has launched liquid staking solutions on Terra, Fantom, Polygon, Hedera, and Near while also developing a thriving DeFi ecosystem to complement their staking solutions. The audited StakeManager contract moves the staking ecosystem on BNB chain to the next level, which supports users to stake their BNB with Stader and receive a synthetic token (BNBx). Stader pools the staked BNB together and optimally stakes it to validator nodes (balancing diversification, APR and fee). The rewards generated from staking are added back to the pool and the value of BNBx (in terms of BNB) increases. The basic information of the audited protocol is as follows:

Table 1.1: Basic Information of The Stader Protocol

Item	Description
Issuer	Stader
Website	https://staderlabs.com/
Type	EVM Smart Contract
Platform	Solidity
Audit Method	Whitebox
Latest Audit Report	July 24, 2022

In the following, we show the Git repository of reviewed files and the commit hash value used in this audit.

- https://github.com/stader-labs/bnbX.git (d56ab58)

And here is the commit ID after all fixes for the issues found in the audit have been checked in:

- https://github.com/stader-labs/bnbX.git (241b7b8)

1.2 About PeckShield

PeckShield Inc. [8] is a leading blockchain security company with the goal of elevating the security, privacy, and usability of current blockchain ecosystems by offering top-notch, industry-leading services and products (including the service of smart contract auditing). We are reachable at Telegram (https://t.me/peckshield), Twitter (http://twitter.com/peckshield), or Email (contact@peckshield.com).

Table 1.2: Vulnerability Severity Classification

		Likelihood		
		High	*Medium*	*Low*
Impact	*High*	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low

1.3 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [7]:

- Likelihood represents how likely a particular vulnerability is to be uncovered and exploited in the wild;
- Impact measures the technical loss and business damage of a successful attack;
- Severity demonstrates the overall criticality of the risk.

Table 1.3: The Full List of Check Items

Category	Check Item
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	Money-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead Of Transfer
	Costly Loop
	(Unsafe) Use Of Untrusted Libraries
	(Unsafe) Use Of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses
Semantic Consistency Checks	Semantic Consistency Checks
Advanced DeFi Scrutiny	Business Logics Review
	Functionality Checks
	Authentication Management
	Access Control & Authorization
	Oracle Security
	Digital Asset Escrow
	Kill-Switch Mechanism
	Operation Trails & Event Generation
	ERC20 Idiosyncrasies Handling
	Frontend-Contract Integration
	Deployment Consistency
	Holistic Risk Management
Additional Recommendations	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

Likelihood and impact are categorized into three ratings: *H*, *M* and *L*, i.e., *high*, *medium* and *low* respectively. Severity is determined by likelihood and impact and can be classified into four categories accordingly, i.e., *Critical*, *High*, *Medium*, *Low* shown in Table 1.2.

To evaluate the risk, we go through a list of check items and each would be labeled with a severity category. For one check item, if our tool or analysis does not identify any issue, the contract is considered safe regarding the check item. For any discovered issue, we might further deploy contracts on our private testnet and run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.3.

In particular, we perform the audit according to the following procedure:

- Basic Coding Bugs: We first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.

- Semantic Consistency Checks: We then manually check the logic of implemented smart contracts and compare with the description in the white paper.

- Advanced DeFi Scrutiny: We further review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

- Additional Recommendations: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

To better describe each issue we identified, we categorize the findings with Common Weakness Enumeration (CWE-699) [6], which is a community-developed list of software weakness types to better delineate and organize weaknesses around concepts frequently encountered in software development. Though some categories used in CWE-699 may not be relevant in smart contracts, we use the CWE categories in Table 1.4 to classify our findings.

1.4 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release, and does not give any warranties on finding all possible security issues of the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contract(s). Last but not least, this security audit should not be used as investment advice.

Table 1.4: Common Weakness Enumeration (CWE) Classifications Used in This Audit

Category	Summary
Configuration	Weaknesses in this category are typically introduced during the configuration of the software.
Data Processing Issues	Weaknesses in this category are typically found in functionality that processes data.
Numeric Errors	Weaknesses in this category are related to improper calculation or conversion of numbers.
Security Features	Weaknesses in this category are concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management. (Software security is not security software.)
Time and State	Weaknesses in this category are related to the improper management of time and state in an environment that supports simultaneous or near-simultaneous computation by multiple systems, processes, or threads.
Error Conditions, **Return Values,** **Status Codes**	Weaknesses in this category include weaknesses that occur if a function does not generate the correct return/status code, or if the application does not handle all possible return/status codes that could be generated by a function.
Resource Management	Weaknesses in this category are related to improper management of system resources.
Behavioral Issues	Weaknesses in this category are related to unexpected behaviors from code that an application uses.
Business Logics	Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application.
Initialization and Cleanup	Weaknesses in this category occur in behaviors that are used for initialization and breakdown.
Arguments and Parameters	Weaknesses in this category are related to improper use of arguments or parameters within function calls.
Expression Issues	Weaknesses in this category are related to incorrectly written expressions within code.
Coding Practices	Weaknesses in this category are related to coding practices that are deemed unsafe and increase the chances that an exploitable vulnerability will be present in the application. They may not directly introduce a vulnerability, but indicate the product has not been carefully developed or maintained.

2 Findings

2.1 Summary

Here is a summary of our findings after analyzing the design and implementation of the `StakeManager` contract of the `Stader` protocol. During the first phase of our audit, we study the smart contract source code and run our in-house static code analyzer through the codebase. The purpose here is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by our tool. We further manually review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

Severity	# of Findings	
Critical	0	
High	0	
Medium	0	
Low	3	🟦🟦🟦
Informational	0	
Total	3	

We have so far identified a list of potential issues.After further analysis and internal discussion, we determined a few issues of varying severities that need to be brought up and paid more attention to, which are categorized in the above table. More information can be found in the next subsection, and the detailed discussions of each of them are in Section 3.

2.2 Key Findings

Overall, these smart contracts are well-designed and engineered, though the implementation can be improved by resolving the identified issues (shown in Table 2.1), including 3 low-severity vulnerabilities.

Table 2.1: Key Stader StakeManager Audit Findings

ID	Severity	Title	Category	Status
PVE-001	Low	Improved Roles Management in Stader	Coding Practices	Fixed
PVE-002	Low	Suggested Event Generations	Status Codes	Fixed
PVE-003	Low	Trust Issue of Admin Keys	Security Features	Mitigated

Besides recommending specific countermeasures to mitigate these issues, we also emphasize that it is always important to develop necessary risk-control mechanisms and make contingency plans, which may need to be exercised before the mainnet deployment. The risk-control mechanisms need to kick in at the very moment when the contracts are being deployed in mainnet. Please refer to Section 3 for details.

3 | Detailed Results

3.1 Improved Roles Management in Stader

- ID: PVE-001
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: BnbX, StakeManager
- Category: Coding Practices [5]
- CWE subcategory: CWE-1126 [1]

Description

The BnbX contract implements a role-based access control mechanism which is inherited from the openzeppelin AccessControlUpgradeable contract. Each role has an associated admin role that can grant the role to a new member or revoke the role from one member. By default, the admin role is DEFAULT_ADMIN_ROLE. However, it is suggested to set the admin role for each role explicitly.

To elaborate, we show below the code snippet from the BnbX contract. In the initialize() routine, it grants DEFAULT_ADMIN_ROLE to the input _manager. And in the setStakeManager() routine, it grants PREDICATE_ROLE to the new stakeManager. That is to say, the BnbX contract does not explicitly set DEFAULT_ADMIN_ROLE as the admin role of PREDICATE_ROLE. Considering the importance of access control, we strongly suggested to explicitly set DEFAULT_ADMIN_ROLE as the admin role of PREDICATE_ROLE in initialize() by calling _setRoleAdmin().

```
    function initialize(address _manager) external override initializer {
        __AccessControl_init();
        __ERC20_init("Liquid Staking BNB", "BNBx");

        require(_manager != address(0), "zero address provided");

        _setupRole(DEFAULT_ADMIN_ROLE, _manager);
    }

    function setStakeManager(address _address)
    external
    override
    onlyRole(DEFAULT_ADMIN_ROLE)
    {
        require(stakeManager != _address, "Old address == new address");
        require(_address != address(0), "zero address provided");

        _revokeRole(PREDICATE_ROLE, stakeManager);
        stakeManager = _address;
        _setupRole(PREDICATE_ROLE, _address);

        emit SetStakeManager(_address);
    }
```

Listing 3.1: BnbX:: initialize ()

Note the same improvement could be applied in `StakeManager` to set `DEFAULT_ADMIN_ROLE` as the admin role of `BOT`.

Recommendation Improve the above mentioned contracts by explicitly setting admin role.

Status The issue has been fixed in the following commits: `a35c48b` and `0d082a8`.

3.2 Suggested Event Generations

- ID: PVE-002
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: `StakeManager`
- Category: Coding Practices [5]
- CWE subcategory: CWE-563 [3]

Description

In `Ethereum`, the `event` is an indispensable part of a contract and is mainly used to record a variety of runtime dynamics. In particular, when an `event` is emitted, it stores the arguments passed in transaction logs and these logs are made accessible to external analytics and reporting tools. `Events` can be emitted in a number of scenarios. One particular case is when system-wide parameters or settings are being changed. Another case is when tokens are being minted, transferred, or burned.

While examining the events that reflect the `minDelegateThreshold` dynamics in the `setMinDelegateThreshold()` routine, we notice there is a lack of emitting an event to reflect `minDelegateThreshold` changes. To elaborate, we show below the code snippet of the `setMinDelegateThreshold()` routine.

```
    function setMinDelegateThreshold(uint256 _minDelegateThreshold)
        external
        override
        onlyRole(DEFAULT_ADMIN_ROLE)
    {
        require(_minDelegateThreshold > 0, "Invalid Threshold");
        minDelegateThreshold = _minDelegateThreshold;
    }
```

Listing 3.2: `setMinDelegateThreshold()`

With that, we suggest to add a new event `SetMinDelegateThreshold()` whenever the `minDelegateThreshold` is updated. Note the same event could be emitted in `initialize()` where the `minDelegateThreshold` is initialized.

What is more, the contract defines `SetBotAddress()/SetBCDepositWallet()` to update the `bot` and `bcDepositWallet`. But there is a lack of emitting these two events in `initialize()` where they are initialized.

```
    function initialize(
        address _bnbX,
        address _manager,
        address _tokenHub,
        address _bcDepositWallet,
        address _bot
    ) external override initializer {
        __AccessControl_init();
        __Pausable_init();

        require(
            ((_bnbX != address(0)) &&
                (_manager != address(0)) &&
                (_tokenHub != address(0)) &&
                (_bcDepositWallet != address(0)) &&
                (_bot != address(0))),
            "zero address provided"
        );

        _setupRole(DEFAULT_ADMIN_ROLE, _manager);
        _setupRole(BOT, _bot);

        bnbX = _bnbX;
        tokenHub = _tokenHub;
        bcDepositWallet = _bcDepositWallet;
        bot = _bot;
        minDelegateThreshold = 1e18;
    }
```

Listing 3.3: `initialize()`

Recommendation Properly emit the above-mentioned events with accurate information to timely reflect state changes. This is very helpful for external analytics and reporting tools.

Status The issue has been fixed in the following commit: `0d082a8`.

3.3 Trust Issue of Admin Keys

- ID: PVE-003
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: StakeManager, BnbX
- Category: Security Features [4]
- CWE subcategory: CWE-287 [2]

Description

In the Stader protocol, there is a privileged manager account (with DEFAULT_ADMIN_ROLE) that plays a critical role in governing and regulating the system-wide operations (e.g., grant new BOT/PREDICATE, set new deposit bot wallet, etc.). Our analysis shows that the privileged account needs to be scrutinized. In the following, we show the representative functions potentially affected by the privileges of the privileged account.

Specifically, the privileged functions in the StakeManager contract allow for the manager to grant new BOT, set new bcDepositWallet address which receives users staking funds in the Beacon chain, and pause/unpause the protocol, etc.

```
    function setBotAddress(address _address)
        external
        override
        onlyRole(DEFAULT_ADMIN_ROLE)
    {
        require(bot != _address, "Old address == new address");
        require(_address != address(0), "zero address provided");

        _revokeRole(BOT, bot);
        bot = _address;
        _setupRole(BOT, _address);

        emit SetBotAddress(_address);
    }

    /// @param _address - Beck32 decoding of Address of deposit Bot Wallet on Beacon
        Chain with ‘0x‘ prefix
    function setBCDepositWallet(address _address)
        external
        override
        onlyRole(DEFAULT_ADMIN_ROLE)
    {
        require(bcDepositWallet != _address, "Old address == new address");
        require(_address != address(0), "zero address provided");

        bcDepositWallet = _address;

        emit SetBCDepositWallet(_address);
    }
```

Listing 3.4: Example Privileged Operations in StakeManager.sol

Moreover, the privileged functions in the BnbX contract allow for the manager to grant new PREDICATE_ROLE which can mint/burn users BnbX token, etc.

```
    function setStakeManager(address _address)
    external
    override
    onlyRole(DEFAULT_ADMIN_ROLE)
    {
        require(stakeManager != _address, "Old address == new address");
        require(_address != address(0), "zero address provided");

        _revokeRole(PREDICATE_ROLE, stakeManager);
        stakeManager = _address;
        _setupRole(PREDICATE_ROLE, _address);

        emit SetStakeManager(_address);
    }
```

Listing 3.5: Example Privileged Operations in BnbX.sol

We understand the need of the privileged functions for proper contract operations, but at the same time the extra power to the privileged account may also be a counter-party risk to the contract users. Therefore, we list this concern as an issue here from the audit perspective and highly recommend making these privileges explicit or raising necessary awareness among protocol users.

Recommendation Promptly transfer the privileged account to the intended DAO-like governance contract. All changes to privileged operations may need to be mediated with necessary timelocks. Eventually, activate the normal on-chain community-based governance life-cycle and ensure the intended trustless nature and high-quality distributed governance.

Status The issue has been acknowledged by the Stader team and derisking measures are put in place by transferring the admin key to a multi-sig account with signers including Stader, protocols on BNB, BNB foundation, investors and reputed community members. Meanwhile the privileges will be made transparent to protocol users.

4 Conclusion

In this audit, we have analyzed the design and implementation of the Stader StakeManager contract. Stader specializes in non-custodial staking solutions for retail, enterprise customers, funds and blockchains. Currently, Stader has launched liquid staking solutions on Terra, Fantom, Polygon , Hedera, and Near while also developing a thriving DeFi ecosystem to complement their staking solutions. The audited StakeManager contract moves the staking ecosystem on BNB chain to the next level, which supports uses to stake their BNB with Stader and receive a synthetic token (BNBx). Stader pools the staked BNB together and optimally stakes it to validator nodes (balancing diversification, APR and fee). The rewards generated from staking are added back to the pool and the value of BNBx (in terms of BNB) increases. The current code base is clearly organized and those identified issues are promptly confirmed and fixed.

Meanwhile, we need to emphasize that Solidity-based smart contracts as a whole are still in an early, but exciting stage of development. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

References

[1] MITRE. CWE-1126: Declaration of Variable with Unnecessarily Wide Scope. https://cwe.mitre.org/data/definitions/1126.html.

[2] MITRE. CWE-287: Improper Authentication. https://cwe.mitre.org/data/definitions/287.html.

[3] MITRE. CWE-563: Assignment to Variable without Use. https://cwe.mitre.org/data/definitions/563.html.

[4] MITRE. CWE CATEGORY: 7PK - Security Features. https://cwe.mitre.org/data/definitions/254.html.

[5] MITRE. CWE CATEGORY: Bad Coding Practices. https://cwe.mitre.org/data/definitions/1006.html.

[6] MITRE. CWE VIEW: Development Concepts. https://cwe.mitre.org/data/definitions/699.html.

[7] OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.

[8] PeckShield. PeckShield Inc. https://www.peckshield.com.