BLOCKSEC

Security Audit

Report for puffer-contracts-updates-private

Date: October 12, 2026 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Chapter 1 Introduction **1**
1.1 About the Audit Target . . . 1
1.2 Disclaimer . . . 2
1.3 Procedure of Auditing . . . 2
1.3.1 Security Issues . . . 2
1.3.2 Additional Recommendation . . . 3
1.4 Security Model . . . 3

Chapter 2 Findings **5**
2.1 Recommendation . . . 5
2.1.1 Align the comment with the implementation . . . 5
2.2 Note . . . 6
2.2.1 Cross-chain messages must be settled before the upgrade . . . 6
2.2.2 Potential centralization risks . . . 6

Report Manifest

Item	Description
Client	Puffer
Target	puffer-contracts-updates-private

Version History

Version	Date	Description
1.0	October 12, 2026	First release

Signature



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

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2026-10-13 03:05:50+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 the Audit Target

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

The audit target (hereinafter referred to as the Target) of this audit is the code repository [1] of puffer-contracts-updates-private of Puffer.

The upgrade of puffer-contracts-updates-private migrates the node operator reward claims from Base to Ethereum Mainnet. To support this migration, the contract L1RewardManager stores reward intervals and Merkle roots, verifies claims, and transfers pufETH directly to the accounts specified in those claims. Each interval has a configurable claiming delay, during which the newest interval can be frozen for subsequent reversal. Frozen intervals can subsequently be reverted, burning the minted pufETH and rolling back reward accounting. The upgrade also updates storage and permissions, removes legacy bridging functionality, and leaves historical rewards on Base in claim-only mode.

Note this audit only focuses on the codes located in the following directories/files.

- mainnet-contracts/src/L1RewardManager.sol
- mainnet-contracts/src/L1RewardManagerStorage.sol
- mainnet-contracts/src/interface/IL1RewardManager.sol

Other files are not within the scope of the audit. Additionally, all dependencies of the Target are considered reliable in terms of both functionality and security, and are therefore not included in the audit scope.

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. Code prior to and including the baseline version (`Version 0`), where applicable, is outside the scope of this audit and assumes to be reliable and secure.

Project	Version	Commit Hash
puffer-contracts-updates-private	`Version 0`	`380600060cd231fd8616ba167e674d4140486dbb`
	`Version 1`	`ce1cd7f371cc769377fe2ee5344313e88c0cd9a8`
	`Version 2`	`0622dd720443e52abead8da62dfffce804537459`

[1] https://github.com/PufferFinance/puffer-contracts-updates-private

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 Target, 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 the Target.

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.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan the Target with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of the Target 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 Security Issues

* Access control
* Permission management
* Whitelist and blacklist mechanisms
* Initialization consistency
* Improper use of the proxy system
* Reentrancy
* Denial of Service (DoS)
* Untrusted external call and control flow
* Exception handling
* Data handling and flow
* Events operation
* Error-prone randomness

- Oracle security
- Business logic correctness
- Semantic and functional consistency
- Emergency mechanism
- Economic and incentive impact

1.3.2 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]. 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 five categories:

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

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

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

- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Partially Fixed** The item has been confirmed and partially fixed by the client.
- **Fixed** The item has been confirmed and fixed by the client.

Chapter 2 Findings

In total, we have **one** recommendation and **two** notes.

- Recommendation: 1
- Note: 2

ID	Severity	Description	Category	Status
1	-	Align the comment with the implementation	Recommendation	Fixed
2	-	Cross-chain messages must be settled before the upgrade	Note	-
3	-	Potential centralization risks	Note	-

The details are provided in the following sections.

2.1 Recommendation

2.1.1 Align the comment with the implementation

Status Fixed in `Version 2`

Introduced by `Version 1`

Description Several NatSpec comments in interface `IL1RewardManager` do not fully reflect the configured permissions or runtime behavior:

1. The NatSpec of function `freezeInterval()` states that the function is restricted to `ROLE_ID_DAO`. However, the migration script assigns its selector to `ROLE_ID_REWARD_WATCHER` and grants this role to a separate watcher account. This inconsistency may mislead integrators about which role is authorized to freeze intervals.

2. The NatSpec of function `isClaimingLocked()` states that an interval is locked only when it does not exist, is frozen, is reverted, or remains within the claiming delay. However, a `false` return value only indicates that the interval-level conditions checked by this function have passed. A subsequent claim may still fail because of other contract-level conditions, such as the pause state or access-control configuration.

```
57 /**
58  * @notice Freezes the claiming of the newest interval. It is possible only during the claiming
       delay.
59  * A frozen interval can only be reverted.
60  * @dev Restricted to ‘ROLE_ID_DAO‘
61  */
62 function freezeInterval(uint256 startEpoch, uint256 endEpoch) external;
```

Listing 2.1: mainnet-contracts/src/interface/IL1RewardManager.sol

```
58     bytes4[] memory watcherSelectors = new bytes4[](1);
59     watcherSelectors[0] = L1RewardManager.freezeInterval.selector;
60
61     calldatas[2] = abi.encodeWithSelector(
```

```
62          AccessManager.setTargetFunctionRole.selector, l1RewardManagerProxy, watcherSelectors,
                ROLE_ID_REWARD_WATCHER
63      );
```

Listing 2.2: mainnet-contracts/script/AccessManagerMigrations/03_GenerateRewardManagerCalldata.s.sol

```
91  /**
92   * @notice Returns `true` if the rewards of the interval can not be claimed at the moment
93   * @dev An interval that does not exist, is frozen, is reverted or is still in its claiming delay
          is locked
94   */
95  function isClaimingLocked(bytes32 intervalId) external view returns (bool);
```

Listing 2.3: mainnet-contracts/src/interface/IL1RewardManager.sol

```
306  function isClaimingLocked(bytes32 intervalId) external view returns (bool) {
307      RewardManagerStorage storage $ = _getRewardManagerStorage();
308      Interval storage interval = $.intervals[intervalId];
309      return interval.status != IntervalStatus.ACTIVE || block.timestamp < interval.claimableAt;
310  }
```

Listing 2.4: mainnet-contracts/src/L1RewardManager.sol

Suggestion Update the comments of the functions `freezeInterval()` and `isClaimingLocked()`.

2.2 Note

2.2.1 Cross-chain messages must be settled before the upgrade

Introduced by Version 1

Description The upgrade script puts the contract `L2RewardManager` on Base into claim-only mode before disabling the legacy bridging functions in contract `L1RewardManager` on Ethereum. As a result, pending mint or reverse operations may deliver tokens without completing their compose callbacks, while pending claimer updates may fail to take effect. Before the upgrade, the project must confirm that all cross-chain messages have been fully settled, no interval remains frozen, and the claiming delay for the last bridged interval has elapsed.

2.2.2 Potential centralization risks

Introduced by Version 1

Description In this project, several privileged roles (e.g., `ADMIN_ROLE`, `ROLE_ID_DAO`, `ROLE_ID_REWARD_WATCHER`, and `ROLE_ID_REWARDS_MINTER`) can conduct sensitive operations, which introduces potential centralization risks. For example, `ADMIN_ROLE` can upgrade the implementation and reconfigure access control, `ROLE_ID_REWARD_WATCHER` can freeze intervals, `ROLE_ID_DAO` can change reward parameters and revert intervals, and `ROLE_ID_REWARDS_MINTER` can publish reward roots and mint allocations. If the private keys of the privileged accounts are lost or maliciously exploited, it could pose a significant risk to the protocol.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS