BLOCKSEC

Security Audit
Report for Puffer Token Contract

Date: September 25, 2024 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Puffer Finance
Target	Puffer Token Contract

Version History

Version	Date	Description
1.0	September 25, 2024	First release

Signature

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

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2024-09-26 06:55:44+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 Puffer Token Contract [1] of Puffer Finance. Contract PUFFER serves as an ERC20 token empowered with pausing and whitelist features.

```
1 mainnet-contracts/src/PUFFER.sol
```

Listing 1.1: Audit Scope for this Report

Other files are not within the scope of this audit. Additionally, all dependencies of the smart contracts within the audit scope are considered reliable in terms of 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.

Project	Version	Commit Hash
Puffer Token Contract	Version 1	c682d80410d4ad060c3788bab7cc6997190dd403

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.

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

1.3 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

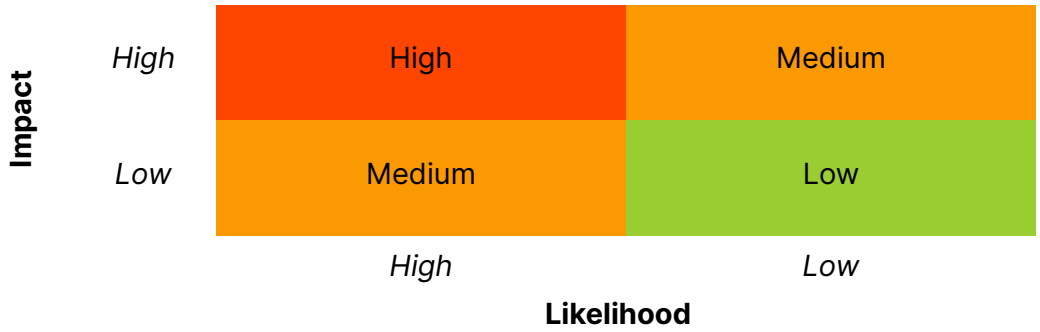
Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

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 [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



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.

[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.
- **Fixed** The item has been confirmed and fixed by the client.

Chapter 2 Findings

In total, we did not find potential security issues. Besides, we have **one** recommendation and **one** note.

- Recommendation: 1
- Note: 1

ID	Severity	Description	Category	Status
1	-	Add checks on the `allowed` parameter	Recommendation	Confirmed
2	-	Potential centralization risks	Note	-

The details are provided in the following sections.

2.1 Additional Recommendation

2.1.1 Add checks on the `allowed` parameter

Status Confirmed

Introduced by `Version 1`

Description In the functions `setAllowedTo` and `_setAllowedFrom`, it is recommended to add a check on the `allowed` parameter to ensure the new value is different from the original one. This check could prevent scenarios where passing an unintended parameter causes ineffective configuration.

```
94    function setAllowedTo(address receiver, bool allowed) external onlyOwner {
95        isAllowedTo[receiver] = allowed;
96        emit SetAllowedTo(receiver, allowed);
97    }
```

Listing 2.1: mainnet-contracts/src/PUFFER.sol

```
108   function _setAllowedFrom(address transferrer, bool allowed) internal {
109       isAllowedFrom[transferrer] = allowed;
110       emit SetAllowedFrom(transferrer, allowed);
111   }
```

Listing 2.2: mainnet-contracts/src/PUFFER.sol

Impact N/A

Suggestion Add a check on the `allowed` parameter against the `isAllowedTo` or `isAllowedFrom`.

2.2 Note

2.2.1 Potential centralization risks

Introduced by `Version 1`

Description The protocol carries potential centralization risks:

In the constructor, contract `PUFFER` mints a fixed total supply of 10^9 tokens (decimals = 18) to the initial owner, who is also granted exclusive permission to send tokens. The contract is initially in a paused state, requiring users to be approved by the owner to send or receive tokens. This pause state will persist until the contract is unpaused by the owner. Considering this excessive privileges, the contract ownership should be transferred to a multi-signature wallet with careful management in place.

Feedback from the Project When the contract is deployed, the deployer needs to distribute tokens to vesting contracts, airdrops, etc. Although token transfers are paused, we do not want users to trade their tokens. However, they should be able to lock their tokens in whitelisted contracts. We need to enable the Airdrop contract to distribute the tokens by setting `isAllowedFrom[airdrop]` to true. Additionally, users should be allowed to lock their claimed tokens in specific contracts by setting `isAllowedTo[locker contract]` to true.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS