BOB Token Security Review

Pashov Audit Group

Conducted by: ast3ros, Kurosaki, ZanyBonzy

February 5th 2025 - February 6th 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 **bob-collective/bob-token** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About BOB Token

BOB Token is an upgradeable ERC-20 token with minting, burning, and role-based access control, using UUPS for upgrades and enforcing a maximum supply. It is deployed on BOB Network. BOB is a hybrid Layer-2 powered by Bitcoin and Ethereum.

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 - 8f31e5301de66c00d34426ca23498991fa721662

fixes review commit hash - 8d80ac7a09a7e6a6b42a92ac22c3432e03317a5b

Scope

The following smart contracts were in scope of the audit:

- `BobToken`

Deployment verification

BOB Token deployment has been verified for the address on BOB Network:

- 0xB0BD54846a92b214C04A63B26AD7Dc5e19A60808

7. Executive Summary

Over the course of the security review, ast3ros, Kurosaki, ZanyBonzy engaged with BOB to review BOB Token. In this period of time a total of **1** issues were uncovered.

Protocol Summary

Protocol Name	BOB Token
Repository	https://github.com/bob-collective/bob-token
Date	February 5th 2025 - February 6th 2025
Protocol Type	ERC20 Token

Findings Count

Severity	**Amount**
Low	1
Total Findings	**1**

Summary of Findings

ID	**Title**	**Severity**	**Status**
[L-01]	Token permit signatures cannot be cancelled before expiry	Low	Resolved

8. Findings

8.1. Low Findings

[L-01] Token permit signatures cannot be cancelled before expiry

The BobToken contract implements `ERC20Permit` functionality for gasless approvals, but lacks a mechanism for users to revoke their permit signatures before they expire. Once a user signs a permit, they must wait until the deadline passes to invalidate it, even if they want to cancel the approval for other security reasons.

This is because the contract is based on OpenZeppelin's `ERC20Permit.sol` in which the function to increase nonce does not exist and the `_useNonce` function within `Nonces` is marked internal. It means the current nonce system only increments when a permit is executed, not allowing users to proactively invalidate pending signatures by advancing their nonce.

Consider introducing a public function that signers can directly use to consume their nonce, thereby canceling the signatures.

```
function useNonce() external returns (uint256) {
        return _useNonce(msg.sender);
    }
```