BOB Gateway Security Review

Pashov Audit Group

Conducted by: ast3ros, 0x37, sashik-eth, Parth, llill, Madalad

March 17th 2025 - March 20th 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-gateway** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About BOB Gateway

BOB is a hybrid Layer-2 powered by Bitcoin and Ethereum. The design is such that Bitcoin users can easily onboard to the BOB L2 without previously holding any Ethereum assets. The user coordinates with the trusted relayer to reserve some of the available liquidity, sends BTC on the Bitcoin mainnet and then the relayer can provide a merkle proof to execute a swap on BOB for an ERC20 token.

BOB Gateway is a bridge solution that enables Bitcoin users to seamlessly onboard to the BOB L2 by swapping BTC for Ethereum-based assets (like wBTC or tBTC) via a trusted relayer and smart contracts, without requiring pre-existing ETH holdings.

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 - b6686882f1bb5ea7837c7a38b2b5a0e93755d55b

fixes review commit hash - 5e6f4571933593e07423ea310a7ae7edf0393bfb

Scope

The following smart contracts were in scope of the audit:

- `CommonStructs`
- `OfframpRegistry`

7. Executive Summary

Over the course of the security review, ast3ros, 0x37, sashik-eth, Parth, llill, Madalad engaged with BOB to review BOB Gateway. In this period of time a total of **8** issues were uncovered.

Protocol Summary

Protocol Name	BOB Gateway
Repository	https://github.com/bob-collective/bob-gateway
Date	March 17th 2025 - March 20th 2025
Protocol Type	Hybrid Bitcoin Layer 2

Findings Count

Severity	**Amount**
Medium	1
Low	7
Total Findings	**8**

Summary of Findings

ID	Title	Severity	Status
[M-01]	Tokens sent to old cached receiver address in OfframpRegistry	Medium	Resolved
[L-01]	Offramp providers can receive ERC20 tokens without sending BTC	Low	Resolved
[L-02]	Offramp cannot submit BTC transactions if submitters inactive	Low	Acknowledged
[L-03]	createOrder() could have the deadline parameter	Low	Resolved
[L-04]	txProofDifficultyFactor should have upper limit	Low	Resolved
[L-05]	Off ramp owners might not retrieve assets from BOB Gateway	Low	Acknowledged
[L-06]	LP cannot prevent emergency offramp fund withdrawals	Low	Resolved
[L-07]	Implementing renounceOwnership() is dangerous	Low	Resolved

8. Findings

8.1. Medium Findings

[M-01] Tokens sent to old `cached receiver address` in `OfframpRegistry`

Severity

Impact: Medium

Likelihood: Medium

Description

In the `OfframpRegistry` contract, when an offramp owner accepts an order, the current receiver address is cached in the order details:

```
function offrampAcceptOrder(uint256 orderId) external {
        ...
        offrampOrder.receiver = offrampDetails.receiver; // update receiver so
        // even if offramp gets deregister current valid orders are process
        ...
    }
```

This is done to ensure orders can still be processed even if the offramp is deregistered. However, this creates a vulnerability when the receiver address is updated before pending orders are processed.

When an offramp owner updates their receiver address via `updateOfframp`. The new receiver address is stored in the offramp details, but it does not update the receiver address in any previously accepted orders.

```
function updateOfframp
      (address token, address newReceiver, uint256 newMinFeeAmount) external {
        ...
        offrampDetails.receiver = newReceiver;
        ...
    }
```

Later, when the BTC transfer is verified and funds are released:

```
function verifyAndReleaseBtcTransfer
      (OfframpProveBtcTransferArgs calldata proveBtcTransferArgs) external {
        ...
        IERC20(offrampOrder.token).safeTransfer
          (offrampOrder.receiver, amountToUnlock);
        ...
    }
```

The tokens are transferred to the cached receiver address (`offrampOrder.receiver`) rather than the updated address in the offramp details.

If a receiver address needs to be updated due to security concerns (e.g., private key compromise) or operational changes, any previously accepted orders will still send tokens to the old, potentially compromised address. This prevents the offramp owner from updating their receiving address for in-flight transactions, which could result in token loss.

Recommendations

In `verifyAndReleaseBtcTransfer`, use the current receiver if available, otherwise fall back to cached receiver.

```
function verifyAndReleaseBtcTransfer
      (OfframpProveBtcTransferArgs calldata proveBtcTransferArgs) external {
        ...

+       if (offramps[offrampOwner][offrampOrder.token].receiver != address(0)) {
+            receiver = offramps[offrampOwner][offrampOrder.token].receiver;
+        } else {
+            receiver = offrampOrder.receiver;
+        }

-        IERC20(offrampOrder.token).safeTransfer
- (offrampOrder.receiver, amountToUnlock);
+        IERC20(offrampOrder.token).safeTransfer(receiver, amountToUnlock);
        emit OrderProcessed(
          proveBtcTransferArgs.orderId,
          amountToUnlock,
          resultInfo.value,
          offrampOrder.token
        );
    }
```

8.2. Low Findings

[L-01] Offramp providers can receive ERC20 tokens without sending BTC

The `verifyAndReleaseBtcTransfer` function in the OfframpRegistry contract has an issue in how it verifies Bitcoin transactions. It only validates two criteria:

- The Bitcoin transaction's output script matches the user's specified Bitcoin address (`outputScript`)
- The transaction value meets or exceeds the minimum required amount (`amountLocked - feesMax`)

However, the contract fails to verify that the transaction was specifically made to fulfill this particular offramp order. This creates a vulnerability where offramp providers can exploit existing unrelated Bitcoin transactions to claim ERC20 tokens without sending BTC specifically for the order.

An offramp provider can:

- Accept a user's order through `offrampAcceptOrder`
- Instead of sending new BTC to fulfill the order, it submits an existing Bitcoin transaction—perhaps one made earlier for a completely different reason (e.g., a business payment or refund)—as long as it:
  - Sends BTC to the same user address (`outputScript`).
  - Has a value greater than or equal to `amountLocked - feesMax`.
- Receive the user's locked ERC20 tokens without actually sending BTC specifically for this order

This allows for double claiming (using the same BTC transaction to fulfill multiple orders).

It's recommended for offramp providers to send UTXO to a specific address that the relayer control (commits to offramp) and that address will forward BTC to users.

[L-02] Offramp cannot submit BTC transactions if submitters inactive

The `verifyAndReleaseBtcTransfer` function has an access control check that allows only authorized actors (submitters) to verify BTC tx on the EVM side. This puts offramp actors into dependent on submitters, so if submitters are inactive or if the owner is compromised and deauthorized all submitters - the offramp cannot finalize BTC tx. At the same time, users could refund their orders on the EVM side so the offramp would face losses.

Consider adding an "emergency verify" function that would allow offramp to verify BTC tx only for their orders in case of time is close to the `CLAIM_DELAY` timestamp. This way offramp could be sure that their trade would be successfully finished and the user would not be able to refund the order while BTC had already transferred.

[L-03] createOrder() could have the deadline parameter

The `createOrder` doesn't have a deadline parameter. Such a feature could be useful for users in case their creation order transaction is stuck in the mempool or they forgot about creating order.

Consider adding an optional `deadline` parameter to the `offrampOrderArgs` struct and check against its value during the `offrampAcceptOrder` call.

[L-04] `txProofDifficultyFactor` should have upper limit

The `setTxProofDifficultyFactor` function has a minimal limit check for the new `txProofDifficultyFactor`, however, it is missing the upper limit check. This could lead to a scenario when the owner accidentally/maliciously sets `txProofDifficultyFactor` to some big value that would prevent BTC transaction verifying calls and would allow users to unlock their funds on the EVM side, while on the BTC side they would receive assets as well.

Consider setting the upper limit check for `txProofDifficultyFactor` to some reasonable value (less than `CLAIM_DELAY` in terms of BTC block time).

[L-05] Off ramp owners might not retrieve assets from `BOB Gateway`

Based on the Bob's documentation(https://docs.gobob.xyz/learn/introduction/roadmap/),`BOB is a Hybrid L2 (Layer 2), a new kind of rollup that inherits Bitcoin security while providing trust-minimized bridges directly to multiple L1s.

Today, BOB has completed Phase 1 of its roadmap. The network is live as an optimistic roll up on Ethereum with a growing TVL of Bitcoin liquid staking tokens (LSTs) in our DeFi ecosystem. Our onchain BTC light client powers trust-minimized cross-chain BTC intents with BOB Gateway.

In current phase, Bob blockchain is one roll up L2 on Ethereum. Like OP/ARB, re-org may happen in this Bob blockchain based on current phase's solution.

If re-org happens, malicious users may monitor this event and try to insert one create order transaction. This may cause the off ramp owner may accept and process one wrong off ramp order.

For example:

1. Alice creates one order for 1 WBTC, the order id is 1.
2. The off ramp owner Bob monitors this order, and accepts this order.
3. Bob sends the related transfer transaction to Bitcoin blockchain and add this BTC transaction via api addBtcTxForOrder/
4. If re-org happens, Alice can insert one order for 1 wei WBTC, the order is id 1.
5. Alice's previous transaction may fail if there is not enough balance in Alice's account.
6. For bob's evm transaction, we still match the order id 1.
7. Based on the case, Alice will get 0.99 BTC in the BTC blockchain, but Bob will lose BTC and can not get the expected WBTC.

Recommendations:

When we identify one off ramp order, we need to use one hash related with all parameters in this order. When off ramp owners accept one order, we will input the related order's hash value.

[L-06] LP cannot prevent emergency offramp fund withdrawals

The `offrampAcceptOrder` function lacks the `whenNotPaused` modifier, creating a vulnerability:

```
function offrampAcceptOrder(uint256 orderId) external {
        ...
    }
```

When the system is paused (likely during an emergency), users should be able to withdraw their locked funds using the `unlockFunds` function. However, the current implementation allows a malicious or compromised LP to accept orders even during a system pause.

When an order moves from Active to Accepted state, it triggers the `CLAIM_DELAY` of 7 days before the user can withdraw:

```
if (offrampOrder.status == OfframpOrderStatus.Accepted) {
    require(
        offrampOrder.timestamp + CLAIM_DELAY < block.timestamp,
        "Funds are still locked and cannot be claimed yet"
    );
}
```

This creates a scenario where:

- The system is paused due to an emergency
- Users attempt to withdraw funds via `unlockFunds`
- A compromised LP rapidly accepts all pending orders
- Users are forced to wait 7 days (per `CLAIM_DELAY`) before they can withdraw
- This effectively denies the opponity to withdraw assets from system. It can lead to loss of fund.

Add the `whenNotPaused` modifier to the offrampAcceptOrder function:

```diff
- function offrampAcceptOrder(uint256 orderId) external {
+ function offrampAcceptOrder(uint256 orderId) external whenNotPaused {
```

[L-07] Implementing `renounceOwnership()` is dangerous

`OfframpRegistry.sol` inherits `Ownable2Step`, and renouncing ownership via `renounceOwnership()` is possible since the function is not overridden. Leaving the contract without an owner would prevent offramps from being added/removed, as well as submitters from being authorized/deauthorized, which can lead to numerous undesirable outcomes:

- Malicious/Compromised offramp can grief users by accepting every order without the intention of processing it by sending BTC, locking user funds for 7 days whenever they create an order
- If the relay contract responsible for calling `verifyAndReleaseBtcTransfer()` becomes compromised or experiences a bug leading to downtime, the contract would be bricked since it would no longer be possible to deauthorize the faulty contract and reauthorize a new one
- `OfframpRegistry.sol` also inherits `Pausable`, meaning that it is possible to renounce ownership while the contract is paused, which would make `createOrder()` permanently uncallable, bricking the contract

Consider overriding `renounceOwnership()` to revert when called by placing the following code in `OfframpRegistry.sol`:

```solidity
function renounceOwnership() public override onlyOwner {
        revert("renouncing ownership not allowed");
    }
```