BOB Gateway Security Review

Pashov Audit Group

Conducted by: juancito, sashik-eth, ZanyBonzy

September 5th - September 7th

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.

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 - c26afe6b0c33e82c828de999dbb5b4279d065f8e

fixes review commit hash - 86c81ac6ecf3060f70f66979ec4a3cd8e0ad4f1d

Scope

The following smart contracts were in scope of the audit:

- `CommonStructs`
- `Constants`
- `Gateway`
- `GatewayRegistry`
- `GatewayRegistryV2`
- `BedrockStrategy`
- `PellStrategy`
- `SegmentStrategy`
- `ShoebillStrategy`
- `SolvStrategy`

7. Executive Summary

Over the course of the security review, juancito, sashik-eth, ZanyBonzy engaged with BOB to review BOB Gateway. In this period of time a total of **6** issues were uncovered.

Protocol Summary

Protocol Name	BOB Gateway
Repository	https://github.com/bob-collective/bob-gateway
Date	September 5th - September 7th
Protocol Type	Hybrid Layer 2

Findings Count

Severity	**Amount**
Low	6
Total Findings	**6**

Summary of Findings

ID	Title	Severity	Status
[L-01]	PellStrategy#stakerStrategyShares function should be view	Low	Resolved
[L-02]	Missing slippage for strategies	Low	Resolved
[L-03]	Missing ETH swap slippage	Low	Acknowledged
[L-04]	Transactions may be stuck for certain recipients	Low	Acknowledged
[L-05]	Inconsistent use of ETH transfer gas limit parameter	Low	Acknowledged
[L-06]	Gas bomb via provided extra data	Low	Resolved

8. Findings

8.1. Low Findings

[L-01] `PellStrategy#stakerStrategyShares` function should be `view`

Currently `PellStrategy#stakerStrategyShares` doesn't have a `view` modifier since it's calling the external non-view function `stakerStrategyShares` from the `IPellStrategyManager` interface:

```
File: PellStrategy.sol
interface IPellStrategyManager {
...
    function stakerStrategyShares(address staker, IPellStrategy strategy) external returns (uint256);
}
...
contract PellStrategy is IStrategy, Context {
...
    function stakerStrategyShares(address recipient) external returns (uint256) {
        return pellStrategyManager.stakerStrategyShares(recipient, pellStrategy);
    }
```

However, in the current implementation of the `pellStrategyManager` the `stakerStrategyShares` is a `view` function: https://explorer.gobob.xyz/address/0x8f083EaFcbba2e126AD9757639c3A1E25a061A08?tab=contract

This means that `PellStrategy#stakerStrategyShares` could be safely restricted to `view`, this would simplify on-chain testing of the contract and allow external integrators, like data collectors, to get the contract's info without the need to send transactions.

[L-02] Missing slippage for strategies

Strategies don't implement any slippage check.

Some strategies might return fewer assets than the user would expect when sending the Bitcoin transaction with the bridge information. This can be due to market changes during the bridging process or sandwich attacks. In either of those scenarios, users might receive fewer assets than expected.

Note: Certain strategies may always return a linear equivalent of tokens (like Bedrock), but that's not necessarily the case for all, and some might depend on external actions from the protocol or other users, affecting the outcome.

Consider getting the expected returned assets from a strategy and setting an appropriate slippage when necessary. This value should be added to the hashed data on the Bitcoin transaction (inside `strategyExtraData` for example), to be decoded later to be applied as a `minAmountOut` comparison with the returned assets.

[L-03] Missing ETH swap slippage

The `releaseFundsAndInvest()` function in `Gateway` provides a `minAmountOut` of zero ETH on swaps. This can lead to the user receiving less ETH than they expected.

```
swapper.swapExactTokensForNative(token, cappedAmountToSwap, 0, payable
  (this), new bytes(0))
```

Note: This has been mentioned in a previous audit, but it was acknowledged considering non-profitable sandwich attacks, given that the amount to be swapped would be very low.

Given that there is no on-chain check to prevent users from swapping large amounts, and that ETH/BTC price fluctuations can happen during the bridging process (without the need for any sandwich attack), it is important to note that users may not receive an ETH amount that is acceptable to them.

Consider adding a slippage parameter for ETH swaps. If doing so, that could be included in the `gatewayExtraData`, which should be hashed to be included in the BTC transaction.

Note for upgrades: Be mindful that pending BTC bridge transactions may not be able to be relayed if not compatible with a new hashing format.

[L-04] Transactions may be stuck for certain recipients

The `Gateway` contract can swap tokens for ETH when relaying a transaction. It uses a try/catch mechanism to prevent the transaction from being stuck if the swapper fails.

The problem is that the relayed transaction can still fail on the `.call` interaction since the expected gas to be set for it is 2300.

Certain recipients, like smart wallets, may have fallback functions that require more than 2300 gas to be executed, and in those cases, the transaction will always revert.

Subsequent attempts would also fail, and funds from the relayed transaction might remain stuck.

```
// After transferred back to the contract, the gas will be limited to 2300
    // by the transfer function before forwarding it to the
    // user supplied address. Doing the transfer directly from the AMM might not
    // limit the gas which could cause reentrancy attack.
    ...

    try swapper.swapExactTokensForNative(token, cappedAmountToSwap, 0, payable
      (this), new bytes(0)) returns (
        uint256 outAmount
    ) {
        receivedEthFromSwap = outAmount;
        (bool sent,) = _proveBtcTransferArgs.recipient.call{
            value: receivedEthFromSwap,
            gas: _proveBtcTransferArgs.ethTransferGasLimit
        }("");
@>      require(sent, "Could not transfer ETH");

        // now subtract the amount that was swapped
        scaledAmountToSendToUser -= cappedAmountToSwap;
    } catch {
        emit EthSwapFailed();
        IERC20(token).safeDecreaseAllowance(address
          (swapper), cappedAmountToSwap);
    }
```

Note: The ETH swap function is expected to swap small amounts of BTC tokens to ETH for an EOA to have gas to operate. However, there is no on-chain limitation that prevents a user from swapping a larger amount and sending it to their smart wallet or another contract. So, a mitigation for those cases would be suggested.

One possible solution would be that the function in the `try` statement, tries to swap the tokens + send them with the expected gas limit altogether.

This way if the operation fails, the users would still receive the BTC tokens to their recipient address, and the transaction would be relayed.

Note: This won't provide the ETH to the user (although it provides the same mitigation as if the swap failed)

[L-05] Inconsistent use of ETH transfer gas limit parameter

The `ethTransferGasLimit` parameter is passed to the `Gateway` as part of the `proveBtcTransferArgs` to limit the gas spent and to prevent any possible re-entrancy attacks.

This value is facilitated by the Relayer via a call to `proveBtcTransfer()` on the `GatewayRegistry`.

If the value is too high, the relayer will spend more on gas and it opens up a re-entrancy opportunity. If the value is too low, the transaction will revert as it can't send the ETH. So it's crucial to set an appropriate value.

It is also worth noting that an integrity check has been added to the BTC transfer arguments provided by users to be relayed, but the `ethTransferGasLimit` value is not included in it.

Considering all of the previous facts, there are two possible issues:

- If the `ethTransferGasLimit` is arbitrarily set by the relayer, it should not belong to the `proveBtcTransferArgs`, as it is not part of the hashed value coming from the BTC transaction. It can also lead to the possibility of configuration errors that would block user transactions.
- If the `ethTransferGasLimit` is provided by the user and blindly relayed by the relayer, that opens up the previously mentioned attack vectors for anyone.

Recommendation

Two approaches can be suggested depending on who is expected to set the `ethTransferGasLimit` value.

If the value is expected to be provided by the protocol and to be fixed, a constant or configurable value would be recommended, as it prevents any error on the relayer side, even if compromised. It would also be suggested to move the `ethTransferGasLimit` value outside of the `proveBtcTransferArgs` struct, but given that the current function interface would change, a comment would be sufficient to prevent compatibility issues.

If the value is expected to be provided by users and later relayed, extra measures should be taken. Minimum and maximum values should be checked (considering extra costs for the relayer if fees are sufficient). The value should also be included in the integrity check (and added to the Bitcoin transaction hashed data).

[L-06] Gas bomb via provided extra data

Users can provide `gatewayExtraData` and `strategyExtraData` values to be relayed so their transactions can be bridged. There is no extra cost for them on the Bitcoin chain regardless of the length of those values, as the data will be hashed first and it would cost the same if it were empty, or if it was huge.

The problem is that the relayer will have to pay extra gas to relay the transaction with large calldata values even if those values are not used.

An adversary can abuse this to create gas bombs for the relayer.

Require that the length of both `gatewayExtraData` and `strategyExtraData` are below a certain reasonable limit for their expected use, considering the minimum expected fees.

This value might be configurable or constant, but it has to be enough for any possible legit scenario, as it would block users' transactions otherwise.