BOB Ordinals Security Review

Pashov Audit Group

Conducted by: carrotsmuggler, Dan Ogurtsov, Alex Murphy

June 24th 2024 - June 26th 2024

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-ordinals-bridge** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About BOB Ordinals Bridge

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. BOB implements Ordinals. Ordinals are a system for tracking and transferring satoshis, Bitcoin's smallest units and attaching data to satoshis. When such attachment of data to satoshi happens an inscription is created. Inscription content is entirely on-chain, stored in taproot script-path spend scripts forever.

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 - 96762a2866edf1f1371bacc527d1a56031b872e2

fixes review commit hash - 70c9cf645b432ad9d5e40a6de5b0970b89b4ae26

Scope

The following smart contracts were in scope of the audit:

- `OrdinalsNFT`
- `Bridge`
- `MultiSig`
- `SigCollection`

7. Executive Summary

Over the course of the security review, carrotsmuggler, Dan Ogurtsov, Alex Murphy engaged with BOB to review BOB Ordinals Bridge. In this period of time a total of **4** issues were uncovered.

Protocol Summary

Protocol Name	BOB Ordinals Bridge
Repository	https://github.com/bob-collective/bob-ordinals-bridge
Date	June 24th 2024 - June 26th 2024
Protocol Type	Hybrid Layer 2

Findings Count

Severity	**Amount**
Medium	1
Low	3
Total Findings	**4**

Summary of Findings

ID	Title	Severity	Status
[M-01]	Gas Price can change, making sigFees calculation insufficient	Medium	Resolved
[L-01]	multisigHash should have an expiration deadline	Low	Resolved
[L-02]	Setting max amount of msg.value for redeem/mint fees	Low	Resolved
[L-03]	Hardcoded BTC_SIG_GAS_USAGE can brick the bridge in case of a network upgrade	Low	Resolved

8. Findings

8.1. Medium Findings

[M-01] Gas Price can change, making `sigFees` calculation insufficient

Severity

Impact: Medium

Likelihood: Medium

Description

When a user asks for a redemption, the validators submit signatures which eventually transfer out the bitcoin ordinal. To compensate the validator's gas fees, the user is required to pay for them.

```
uint256 sigFees = BTC_SIG_GAS_USAGE * tx.gasprice * uint256
  (multisig.activeCommitteeSize);
```

This `sigFees` is meant to cover the gas cost of the validators, which is incurred by them when they post a multisig signature. The issue is that this function uses `tx.gasPrice`. This is a user controlled argument, and heavily depends on the network traffic.

Say Alice wants to redeem her ordinal. She calls `redeem` during a time of low network traffic, so she sets her `tx.gasPrice` as 0.1 gwei. When the validators are required to submit the signatures, the gas price might have shot up due to higher network traffic. The validators are then forced to use a higher gas price, say 0.2 gwei to get their transaction through.

Since only 0.1 gwei priced gas fees were set aside, the validator will simply refuse to do the transaction.

This is not Alice's fault, since she probably did the transaction with a wallet that guesses the optimum gas price to set for the current block. If Alice wants to make sure her redemption goes through, she needs to manually set her `tx.gasPrice` higher, anticipating future higher network usage. However, this causes her to waste gas, since she is now bidding at a much higher gas price for the current transaction, which doesn't need it.

In addition, it is possible to avoid paying fees by setting zero gas prices.

Recommendation

Consider allowing ALICE to specify how much eth she wants to allocate to the validators to cover gas fees, and then reimburse the rest to her after the redemption is over.

8.2. Low Findings

[L-01] multisigHash should have an expiration deadline

Currently, `multisigHash` statements do not have an expiry date. This poses some risks in case of lowering the `threshold` variable or some validators going malicious at some point in time which may cause some previously malicious `multisigHash` to suddenly be approved.

[L-02] Setting max amount of `msg.value` for redeem/mint fees

`msg.value` can be accepted to top up `redeemFees` and `mintFees`. But it can be any number even if it is far above the required amount. Consider comparing with some max allowed required amount that would make sense.

[L-03] Hardcoded `BTC_SIG_GAS_USAGE` can brick the bridge in case of a network upgrade

The function `_remFees` in `Bridge.sol` contract deducts gas fees for the validators. It calculates this assuming a fixed cost for submitting the bitcoin signature.

```
uint256 private constant BTC_SIG_GAS_USAGE = 33704;
uint256 sigFees = BTC_SIG_GAS_USAGE * tx.gasprice * uint256
  (multisig.activeCommitteeSize);
```

However, due to a network upgrade, gas fees for different operations like store, read etc can change. If the fee for submitting a bitcoin signature changes, then the `sigFees` calculated will be off, since `BTC_SIG_GAS_USAGE` is defined as a

constant and cannot be changed. This would require a re-deployment of the bridge.

Consider allowing `BTC_SIG_GAS_USAGE` to be changeable.