BOB Security Review

Pashov Audit Group

Conducted by: dirk_y, ast3ros, 0xbepresent

August 9th 2024 - August 12th 2024

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

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 - 1511179bfc908b73020e8c3b668957c7857a8c61

fixes review commit hash - 36b5dec7446538c9e54e94291a755ace4f0cf920

Scope

The following smart contracts were in scope of the audit:

- `OnrampV1`
- `OnrampFactoryV1`
- `Gateway`
- `GatewayRegistry`
- `ERC20Mintable`
- `Constants`
- `CommonStructs`
- `IERC20Ext`
- `IGateway`
- `VelodromeSwapper`
- `TestnetSwapper`
- `ISwapper`
- `IRouter`
- `IWETH`

7. Executive Summary

Over the course of the security review, dirk_y, ast3ros, 0xbepresent engaged with BOB to review BOB. In this period of time a total of **7** issues were uncovered.

Protocol Summary

Protocol Name	BOB
Repository	https://github.com/bob-collective/bob-gateway
Date	August 9th 2024 - August 12th 2024
Protocol Type	Hybrid Layer 2

Findings Count

Severity	**Amount**
Low	7
Total Findings	**7**

Summary of Findings

ID	Title	Severity	Status
[L-01]	Unnecessary onlyOwner restriction	Low	Resolved
[L-02]	Deadline is set to block.timestamp	Low	Acknowledged
[L-03]	Velodrome pool pauses can potentially lead to disrupting BTC bridging	Low	Resolved
[L-04]	The gateway owner can set a malicious swapper that could cause losses	Low	Resolved
[L-05]	Lack of fee range length limit	Low	Acknowledged
[L-06]	High convertedBtcAmountToBeSwappedToEth value setting allowed	Low	Acknowledged
[L-07]	Potential blockage of relayer operations due to token pause mechanism	Low	Acknowledged

8. Findings

8.1. Low Findings

[L-01] Unnecessary onlyOwner restriction

The `depositERC20` function in the Gateway contract is restricted to `onlyOwner`, allowing only the contract owner to deposit tokens into the contract. However, this restriction is largely ineffective as anyone can directly transfer tokens to the contract address, bypassing this function entirely.

```
/**
     * @dev Deposit tokens in the contract.
     * Can only be called by the current owner
     */
    function depositERC20
    //(uint256 amount) external onlyOwner { // @audit restrict to owner but anyone can tra
        emit DepositERC20(amount);
        IERC20(token).safeTransferFrom(_msgSender(), address(this), amount);
    }
```

When getting the liquidity of the Gateway, the relayer calls `availableLiquidity()` directly to query the token balance of the contract.

[L-02] Deadline is set to block.timestamp

In the `swapExactTokensForNative` function, when calling the router to swap tokens for ETH, the deadline parameter is set to block.timestamp. This means the deadline is set to the current block timestamp when the transaction is included by the validator, not when it's submitted. Without a deadline parameter, the transaction may sit in the mempool and be executed at a much later time potentially resulting in a worse price for the user. However, due to the amountIn being a small amount of ETH, the impact is low.

Consider adding a `deadline` parameter to the `swapExactTokensForNative` function, allowing the relayer to set an appropriate future timestamp.

```
function swapExactTokensForNative(
        IERC20 _token,
        uint256 _amountIn,
        uint256 _amountOutMin,
        address payable _to,
        bytes memory
    ) external override returns (uint256 outAmount) {
        ...

        outAmount = router.swapExactTokensForETH
        //(_amountIn, _amountOutMin, routes, _to, block.timestamp)[1]; // @audit deadline
    }
```

[L-03] Velodrome pool pauses can potentially lead to disrupting BTC bridging

The Velodrome pool, which is used for swapping tokens in the gateway process, can be paused by the pool factory admin. If the pool is paused, the swap token for ETH function will revert, preventing BTC from being bridged to the Bob L2 chain despite multiple retry attempts.

```
function swap(
      uint256amount0Out,
      uint256amount1Out,
      addressto,
      bytescalldatadata
    ) external nonReentrant {
        if (IPoolFactory(factory).isPaused()) revert IsPaused();
```

It's recommended to check if the pool is paused, then skip the swap eth for the user.

[L-04] The gateway owner can set a malicious swapper that could cause losses

The `setSwapper` function allows the `Gateway owner` to set any contract as the swapper, If the owner sets a malicious contract as the swapper, the malicious contract can be programmed to steal funds during the swap process:

```
...
    function _setSwapper(ISwapper _swapper) internal {
        emit UpdateSwapper(address(_swapper));
        swapper = _swapper;
    }
...
...
    function releaseFundsAndInvest(
        bytes32 _txHash,
        uint256 _outputValueSat,
        address payable _recipient,
        bool _sendEthToUser,
        uint256 _ethTransferGasLimit,

                bytes memory /* any extra data that is needed for this specific gateway c
    ) external override onlyRegistry {
        ...
        ...
            receivedEthFromSwap =
                swapper.swapExactTokensForNative
                  (token, cappedAmountToSwap, 0, payable(this), new bytes(0));
        ...
        ...
    }
...
...
```

It is recommended that the swap contract only be set at the beginning during the creation of the `Gateway` contract, as this process is audited by the protocol itself, which also pays for it (permissioned). Otherwise, a malicious owner could change the swapper contract, and there may be users who agree to the terms using a malicious swapper.

[L-05] Lack of fee range length limit

In the `Gateway.sol` contract, the `setFeeRanges` function lacks a limit on the number of fee ranges that can be set. Since the protocol pays for the deployments of the `Gateway.sol`

> LP asks relayer to deploy gateway contract (permissioned because we pay for fees)

, an unrestricted number of fee ranges can lead to excessive gas consumption, resulting in higher operational costs for the protocol.

```
constructor
      (GatewayConstructorArgs memory gatewayConstructorArgs) Ownable2Step() {
        require(gatewayConstructorArgs.gatewayOwner != address
          (0), "Owner is the zero address");
        _transferOwnership(gatewayConstructorArgs.gatewayOwner);

        registry = gatewayConstructorArgs.gatewayRegistry;
        token = gatewayConstructorArgs.token;
        require(token.decimals() >= 8, "Invalid token");
        multiplier = 10 ** (token.decimals() - 8);

        _setOutputScript(gatewayConstructorArgs.outputScript);
        _setDustThreshold(_DEFAULT_DUST_THRESHOLD);
        _setConvertedBtcToBeSwappedToEth
          (gatewayConstructorArgs.btcToBeSwappedToEth);
>>>     _setFeeRanges(gatewayConstructorArgs.feeRanges);
        _setSwapper(gatewayConstructorArgs.swapper);
    }
...
...
   function _setFeeRanges(FeeRange[] memory _feeRanges) internal {
        uint256 feeRangesLength = _feeRanges.length;
        require(
          _feeRanges[0].amountLowerRange==0,
          "Firstrangelowerboundaryhastobe0"
        );

>>>     for (uint256 i; i < feeRangesLength - 1; ++i) {
            require(

                                    _feeRanges[i].amountLowerRange < _feeRanges[i + 1].amountl
                "Amount lower ranges need to be sorted ascending"
            );
            require(

                                    _feeRanges[i].scaledFeePercent > _feeRanges[i + 1].scaledl
                "Scaled fee percent need to be sorted descending"
            );
            require(
              _feeRanges[i].scaledFeePercent<=SCALED_MAX_FEE,
              "Feemustbe<=maximumfeeforallranges"
            );
        }
        require(
            _feeRanges[feeRangesLength - 1].scaledFeePercent <= SCALED_MAX_FEE,
            "Fee must be <= maximum fee for all ranges"
        );

        emit UpdateFeeRanges(_feeRanges);

        // Solc compiler: Copying of type struct FeeRange memory[] memory to
        // storage not yet supported.
        delete feeRanges;
        for (uint256 i; i < feeRangesLength; ++i) {
            feeRanges.push(_feeRanges[i]);
        }
    }
...
...
```

Furthermore, each time the relayer calls the `releaseFundsAndInvest` function, the fees are calculated. This gas expenditure is paid by the relayer, thereby increasing the operational costs for the protocol.

```
function releaseFundsAndInvest(
        bytes32 _txHash,
        uint256 _outputValueSat,
        address payable _recipient,
        bool _sendEthToUser,
        uint256 _ethTransferGasLimit,

                bytes memory /* any extra data that is needed for this specific gateway c
    ) external override onlyRegistry {
        ...
        ...
        // Passing feeRanges to calculateFee even though this incurs a small gas
        // penalty compared to directly using it inside the method.
        // Passing it to calculateFee will make the method pure and make testing
        // easier.
>>>     uint256 feeSat = calculateFee(_outputValueSat, feeRanges);
        ...
        ...
    }
...
...
   function calculateFee(
     uint256_amount,
     FeeRange[]memory_feeRanges
   ) public view virtual returns (uint256 fee
        uint256 feeRangesLength = _feeRanges.length;

        uint256 feeRangeIndex = 1;
>>>     for (; feeRangeIndex < feeRangesLength; ++feeRangeIndex) {
            if (_feeRanges[feeRangeIndex].amountLowerRange > _amount) {
                break;
            }
        }
        --feeRangeIndex;

        fee =
          (_amount * _feeRanges[feeRangeIndex].scaledFeePercent) / Constants.FEE_SCALER;
    }
```

Recommendations

It is recommended to impose a reasonable limit on the number of fee ranges that can be set.

[L-06] High `convertedBtcAmountToBeSwappedToEth` value setting allowed

In the `Gateway` contract, the owner has the ability to set the amount of converted BTC that will be swapped to `wETH` via the `Gateway::setConvertedBtcToBeSwappedToEth` function. The swapped `wETH` is then sent to the recipient. The `amountOut` parameter in the swapping function is set to 0, which means the swap will proceed regardless of the amount of ETH received, as mentioned in the code comments, for the swap, `amountOut` is set to zero because it

is not economically valuable as the converted amount is small. This setup is intended to prevent sandwich attacks, but it does not work under certain conditions.

```
function releaseFundsAndInvest(
        bytes32 _txHash,
        uint256 _outputValueSat,
        address payable _recipient,
        bool _sendEthToUser,
        uint256 _ethTransferGasLimit,

                bytes memory /* any extra data that is needed for this specific gateway c
    ) external override onlyRegistry {
        // slither-disable-next-line timestamp
        require(
          updateStart==0||block.timestamp<=updateStart+UPDATE_DELAY,
          "Notallowedtoexecute"
        );

        require(_outputValueSat >= dustThreshold, "Amount too small");

        require(!spent[_txHash], "Transaction already spent");
        spent[_txHash] = true;

        // Passing feeRanges to calculateFee even though this incurs a small gas
        // penalty compared to directly using it inside the method.
        // Passing it to calculateFee will make the method pure and make testing
        // easier.
        uint256 feeSat = calculateFee(_outputValueSat, feeRanges);

        // scale the Btc so the decimal count of Btc will correspond to
        // converted Btc
        uint256 scaledAmount = (_outputValueSat - feeSat) * multiplier;

        // cap at the amount of Btc sent by the user, meaning if the user sends
        // less Btc
        // then the conversion amount everything is swapped to Eth
>>>     uint256 cappedAmountToSwap = Math.min
  (convertedBtcAmountToBeSwappedToEth, scaledAmount);

        uint256 receivedEthFromSwap = 0;
        if (_sendEthToUser) {
            // now subtract the amount to be swapped
            scaledAmount = scaledAmount - cappedAmountToSwap;

            IERC20(token).safeIncreaseAllowance(address
              (swapper), cappedAmountToSwap);
            // 1.
            // For extra safety against contract level reentrancy, the ETH is
            // transferred back to the

            // 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.

>>>         // 2.
>>>         // The amountOut for the swap is set to 0, as the small amount of
// converted Btc will make sandwich attacks
>>>         // not worthy economically to carry out.
>>>         // Denial of service attacks are still possible where an attacker
// would make sandwich attacks to decrase the amount of
>>>         // ETH the user receives to a near 0 amount. It would be costly
// though for the attacker.
>>>         receivedEthFromSwap =
>>>             swapper.swapExactTokensForNative
  (token, cappedAmountToSwap, 0, payable(this), new bytes(0));

            (
              boolsent,

            ) = _recipient.call{value: receivedEthFromSwap, gas: _ethTransferGasLimit}(""
            require(sent, "Could not transfer ETH");
        }
        emit ExecuteSwap(
```

1. The user accepts the terms and transfers `btc` to the `gateway owner's` account (scriptPubKeyHash).
2. The gateway owner initiates `Gateway::startUpdate`, and the relayer has only 6 hours to execute the order generated in step 1.
3. For some reason, the ERC20 `Gateway.token` activates the pause mechanism, and the transfers are reverted.
4. The relayer invokes `GatewayRegistry::proveBtcTransfer`, which in turn calls `Gateway::releaseFundsAndInvest`; however, the token transfer is not possible because the token is paused (the transaction is reverted).

```
function releaseFundsAndInvest(
        bytes32 _txHash,
        uint256 _outputValueSat,
        address payable _recipient,
        bool _sendEthToUser,
        uint256 _ethTransferGasLimit,

                bytes memory /* any extra data that is needed for this specific gateway c
    ) external override onlyRegistry {
        ...
        ...
        // transfer converted Btc token
>>>     IERC20(token).safeTransfer(_recipient, scaledAmount);
    }
```

5. The 6-hour period expires, and now the `Gateway.owner` can make changes to the contract's state, allowing the `Gateway.owner` to withdraw the ERC20 tokens. Meanwhile, the token is unpaused.
6. The relayer attempts to execute `releaseFundsAndInvest` again, but the `Gateway` contract no longer has any tokens available.

WBTC is an example of a token that includes a pause mechanism. This feature allows the administrators of the token to halt transfers temporarily, which can be utilized during upgrades or in response to security threats.

```
contract WBTC is StandardToken, DetailedERC20("Wrapped BTC", "WBTC", 8),
    MintableToken, BurnableToken, PausableToken, OwnableContract {
    ...
    ...
```

It is recommended to implement a pull mechanism to allow the recipient to obtain their corresponding tokens once the token is unpaused. Additionally, the balance should be set aside to prevent the `Gateway.owner` from withdrawing the `ERC20` tokens that are already claimable.

1. The token is paused.
2. The relayer calls `Gateway::releaseFundsAndInvest`, at which point the ERC20 tokens are deducted from the gateway's balance. This action ensures that the

`Gateway.owner` cannot withdraw the ERC20 tokens, as they are earmarked to be retrievable by the recipient as soon as the ERC20 token contract is unpaused.
3. Time passes, the token is unpaused, and the recipient can pull the reserved tokens.