Audit Report

Skip Protocol Owned Builder

v1.0

July 29, 2023

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by Skip Protocol Inc. to perform a security audit of Skip Protocol Owned Builder.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/skip-mev/pob
Commit	`7f448c678ffa020a37d94ec86acca087ae2b3261`
Scope	All code was in scope.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

This audit covers the functionality associated with Skip Protocol's Protocol-Owned Builder. Protocol-Owned Builder (POB) is a set of Cosmos SDK and ABCI++ primitives that provide application developers the ability to define how their apps construct and validate blocks on-chain in a transparent, enforceable way, such as giving complete control to the protocol to recapture, control, and redistribute MEV.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Medium-High	The codebase was very readable and utilized Go and Cosmos SDK best practices.
Level of documentation	Medium-High	The module included extensive documentation.
Test coverage	Medium-High	The builder module included extensive unit testing and was integrated into a test app for end-to-end testing.

Summary of Findings

No	Description	Severity	Status
1	Ante Handler panics if the highest bid denom is not the same as `minBidIncrement`	Minor	Resolved
2	Zero `MinBidIncrement` undermines fee market	Minor	Resolved
3	Parsing query command flags are ignored	Minor	Resolved
4	Unhandled errors in Cobra command	Informational	Resolved
5	Incorrect field numbering	Informational	Resolved
6	Inefficient execution in codebase	Informational	Resolved
7	Identical errors are not combined into variable	Informational	Resolved

Detailed Findings

1. Ante Handler panics if the highest bid denom is not the same as `minBidIncrement`

Severity: Minor

In `x/builder/keeper/auction.go:72`, the highest bid is added with the minimum bid increment without checking if both denoms are identical. This could happen because there are no validations that ensure the submitted bid denom is the same as the configured minimum bid increment denom. Consequently, the Ante Handler will panic without returning an error.

Please see the `TestTwoAuctionsWithDifferentDenom` test case in the Appendix to reproduce the issue.

Recommendation

We recommend validating both denom values and discarding the `highestBid` as an invalid bid transaction if the denom does not match the minimum bid increment denom.

Status: Resolved

2. Zero `MinBidIncrement` undermines fee market

Severity: Minor

The `Params`' `Validate` function in `x/builder/types/params.go:50` does not enforce the `MinBidIncrement` value to be greater than zero. This means that governance or genesis may introduce a value of zero which will undermine the fee market of the builder module.

Consequently, this would create a situation where the module's bid execution may differ from the documentation and confuse searchers. This is because the `validateFee` function performs a basic Cosmos SDK coin validation which allows coins to have a zero amount. While this may slightly modify the behavior of the auction, it ultimately is up to the market actors to place auction bids to ensure their bundle is executed so we only classify this issue as minor.

Recommendation

We recommend explicitly validating that the `MinBidIncrement` is not set to zero in the `Validate` function.

Status: Resolved

3. Parsing query command flags are ignored

Severity: Minor

The `CmdQueryParams` CLI command in `x/builder/client/cli/query.go:31` uses `GetClientContextFromCmd` to get `Context`, yet it does not read query command flags.

Recommendation

We recommend replacing `GetClientContextFromCmd` with `GetClientQueryContext`.

Status: Resolved

4. Unhandled errors in Cobra command

Severity: Informational

In `x/builder/client/cli/tx.go:45`, the return parameter from the `Set` function is not handled. If an error occurs, the `NewAuctionBidTx` function will fail silently.

Recommendation

We recommend handling returned errors.

Status: Resolved

5. Incorrect field numbering

Severity: Informational

In `proto/pob/builder/v1/tx.proto:30`, `MsgAuctionBid` message skips field number 2. The `bid` in line `41` should use 2, and `transactions` in line `45` should use 3 field numbers.

Recommendation

We recommend correcting the field numbers.

Status: Resolved

6. Inefficient execution in codebase

Severity: Informational

In `x/builder/keeper/auction.go:79`, the `GetAllBalances` function is called to get the bidder's account balances to validate that they have sufficient funds to bid. However, it is

inefficient to validate all account balances because the `bid` deals with a single coin denomination. The gas consumption can be reduced by using `GetBalance`, which retrieves one coin balance instead of all coin balances.

Additionally, the `AuctionBid` function in `x/builder/keeper/msg_server.go:38-55` calls the `GetParams` function multiple times. The gas consumption can be reduced by calling `GetParams` once and using the returned value for max bundle size, proposer fee, and the escrow account.

Recommendation

We recommend implementing the optimizations mentioned above.

Status: Resolved

7. Identical errors are not combined into variable

Severity: Informational

In `x/builder/keeper/auction.go:117` and `125`, both returned errors are the same. This decreases maintainability as modifying one error message requires modifying the other one.

Recommendation

We recommend declaring a variable that represents the error message and using that variable instead.

Status: Resolved

Appendix A: Test Cases

1. Test case for "Panic in Ante Handler if the highest bid denom is not the same as `minBidIncrement`"

```
func (suite *AnteTestSuite) TestTwoAuctionsWithDifferentDenom() {
	var (
		// Bid set up
		bidder  = testutils.RandomAccounts(suite.random, 1)[0]
		bid     = sdk.NewCoin("foo", sdk.NewInt(1000))
		balance = sdk.NewCoins(sdk.NewCoin("foo", sdk.NewInt(10000)))
		signers = []testutils.Account{bidder}

		// Top bidding auction tx set up
		topBidder    = testutils.RandomAccounts(suite.random, 1)[0]
		topBid       = sdk.NewCoin("foo", sdk.NewInt(100))
		insertTopBid = true
		timeout      = uint64(1000)

		// Auction setup
		maxBundleSize          uint32 = 5
		reserveFee                    = sdk.NewCoin("foo", sdk.NewInt(100))
		minBidIncrement               = sdk.NewCoin("foo", sdk.NewInt(100))
		frontRunningProtection        = true
	)

	cases := []struct {
		name     string
		malleate func()
		pass     bool
	}{
		{
			/*
				NOTE: If the issue is unpatched but no panic occur, re-run the test case a few more times.

				Expected output:

				panic: invalid coin denominations; bar, foo [recovered]
			*/
			"topBid is another denom && total 2 auction txs",
			func() {
				// set different denom
				topBid.Denom = "bar"
			},
			true,
		},
```

```
	}

	for _, tc := range cases {
		suite.Run(tc.name, func() {
			suite.SetupTest()
			tc.malleate()

			suite.ctx = suite.ctx.WithBlockHeight(1)

			// Set the auction params
			err := suite.builderKeeper.SetParams(suite.ctx,
buildertypes.Params{
				MaxBundleSize:          maxBundleSize,
				ReserveFee:             reserveFee,
				MinBidIncrement:        minBidIncrement,
				FrontRunningProtection: frontRunningProtection,
			})
			suite.Require().NoError(err)

			// Insert the top bid into the mempool
			config :=
mempool.NewDefaultAuctionFactory(suite.encodingConfig.TxConfig.TxDecoder())
			mempool :=
mempool.NewAuctionMempool(suite.encodingConfig.TxConfig.TxDecoder(),
suite.encodingConfig.TxConfig.TxEncoder(), 0, config)
			if insertTopBid {
				topAuctionTx, err :=
testutils.CreateAuctionTxWithSigners(suite.encodingConfig.TxConfig, topBidder,
topBid, 0, timeout, []testutils.Account{})
				suite.Require().NoError(err)
				suite.Require().Equal(0, mempool.CountTx())
				suite.Require().Equal(0, mempool.CountAuctionTx())
				suite.Require().NoError(mempool.Insert(suite.ctx,
topAuctionTx))
				suite.Require().Equal(1, mempool.CountAuctionTx())
			}

			// Create the actual auction tx and insert into the mempool
			auctionTx, err :=
testutils.CreateAuctionTxWithSigners(suite.encodingConfig.TxConfig, bidder, bid,
0, timeout, signers)
			suite.Require().NoError(mempool.Insert(suite.ctx,
auctionTx)) // add bidder tx to mempool
			suite.Require().NoError(err)

			// Execute the ante handler
			suite.builderDecorator =
ante.NewBuilderDecorator(suite.builderKeeper,
suite.encodingConfig.TxConfig.TxEncoder(), mempool)
			_, err = suite.executeAnteHandler(auctionTx, balance)
```

```
                    if tc.pass {
                            suite.Require().NoError(err)
                    } else {
                            suite.Require().Error(err)
                    }
            })
    }
}
```