BURRA SEC

GasbotV2 Security Review

Reviewed by: windhustler

GasbotV2 Security Review Report

Burra Security

Jan 24, 2024

Introduction

A time-boxed security review of the **GasbotV2** protocol was done by **Burra Security** team, focusing on the security aspects of the smart contracts.

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 vulnerabilities. Subsequent security reviews, bug bounty programs, and on-chain monitoring are recommended.

About Burra Security

Burra Sec offers security auditing and advisory services with a special focus on cross-chain and interoperability protocols and their integrations.

About GasbotV2

GasbotV1 is a smart contract that allows depositing stablecoins (like USDC) on any network and receiving gas on another network. The V2 iteration adds the functionality of adding gas tokens in addition to stablecoins to receive gas on the destination network.

Severity classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact - The technical, economic, and reputation damage from a successful attack

Likelihood - The chance that a particular vulnerability gets discovered and exploited

Severity - The overall criticality of the risk

Informational - Findings in this category are recommended changes for improving the structure, usability, and overall effectiveness of the system.

Security Assessment Summary

***review commit hash* - efb5e1d3735f24c7fadb17d59247a262e2647c7b**

***mitigation review commit hash* - 27dcd1c7e6ec87c837cc7b07e4cd966459f75c7d**

Scope

The following smart contracts were in the scope of the audit:

- `src/GasbotV2.sol`

Findings Summary

ID	Title	Severity	Status
[M-01]	`approve` always reverts with tokens missing return value, e.g. USDT	Medium	Resolved

ID	Title	Severity	Status
[M-02]	`swapGas` doesn't work on chains having chainId bigger than `type(uint16).max`	Medium	Resolved
[M-03]	Allow passing deadline check	Medium	Resolved
[L-01]	Missing explicit check that `WETH` is not one of the assets in the `GasBotV2` contract	Low	Resolved
[I-01]	`maxValue` check not enforced in permissioned functions	Info	Ack
[I-02]	Users can be forced to receive gas in case of hanging allowances	Info	Ack
[I-03]	Minimum added value not checked	Info	Resolved

Detailed Findings

[M-01] `approve` always reverts with tokens missing return value, e.g. USDT

Context

- GasbotV2.sol

Description

`USDT` token deployed on Ethereum mainnet does not return any value for `approve` function: https://etherscan.deth.net/token/0xdac17f958d2ee523a2206206994597c13d831ec7.

Using standard `IERC20` interface from OpenZeppelin, `approve` function will always revert with `tokens` missing return value. As `USDT` is highly likely to be used as payment token with GasbotV2, this is marked as high severity issue.

Another part of this issue is that some non-standard tokens like `USDT` will also revert when a contract or a user tries to approve an allowance when the spender's allowance has already been set to a non-zero value. In the current code allowance is set to `UniswapV2` or `UniswapV3` router contracts which will lower the allowance to zero, but if something changes and the allowance is not lowered to 0 then the approval would fail with `USDT`.

I would also note that OpenZeppelin has officially deprecated the `safeApprove` function, suggesting to use `safeIncreaseAllowance` and `safeDecreaseAllowance` instead.

Recommendation

Decrease the allowance to zero before approving it again using `safeDecreaseAllowance` and `safeIncreaseAllowance` functions from OpenZeppelin.

```
-    IERC20(_tokenIn).approve(_router, _amount);
+    uint256 allowance = IERC20(_tokenIn).allowance(address(this), _router);
     if (allowance > 0) {
         IERC20(_tokenIn).safeDecreaseAllowance(_router, allowance);
     }
     IERC20(_tokenIn).safeIncreaseAllowance(_router, _amount);
```

[M-02] swapGas doesn't work on chains having chainId bigger than `type(uint16).max`

Context

- GasbotV2.sol

Description

Some chains such as Scroll have chainId bigger than `type(uint16).max` which will make `swapGas` unusable.

Recommendation

Change `_toChainId` type to `uint256` to support all chains.

```
-    uint16 _toChainId
+    uint256 _toChainId
```

[M-03] Allow passing deadline check

Context

- GasbotV2.sol

Description

`_swap` is an internal function used by `relayTokenIn`, `transferGasOut`, `relayAndTransfer` and `swapGas` functions. It allows to perform a swap through UniswapV2 or UniswapV3 router contracts. Both of these router contracts have the deadline parameter that lets the caller enforce a time limit by which the transaction must be executed.

Passing `block.timestamp` as deadline means that the transaction can be executed at any time. This is an issue since the transaction can get pending for longer periods of time, during which the price of the tokens can change significantly.

Recommendation

Allow passing deadline as a parameter externally to `relayTokenIn`, `transferGasOut`, `relayAndTransfer` and `swapGas` functions.

[L-01] Missing explicit check that `WETH` is not one of the assets in the `GasbotV2` contract

Context

- GasbotV2.sol

Description

To understand why `GasbotV2` contract should not hold `WETH` token, we need to examine the `transferGasOut` and `relayAndTransfer` function. Both of these functions after the swap will transfer all the unwrapped ETH to the user. This happens inside the `_transferAtLeast` function. So the assumption is that `WETH` is only generated after swapping the tokens that the user has provided.

If we look at `relayTokenIn` function, there is no check that `homeToken` is not `WETH`. Also, there is no check that the relayer hasn't swapped the tokenIn for `WETH`.

Recommendation

Add a check that `homeToken` is not `WETH` in contract constructor and the `setHomeToken` function. Check that the balance of `WETH` hasn't changed in `relayTokenIn` function.

[I-01] `maxValue` check not enforced in permissioned functions

Context

- GasbotV2.sol

Description

`maxValue` check is enforced only in the `swapGas` function. Since other functions are permissioned, the assumption is that the server side code checks that the `maxValue` is not exceeded.

Best practice would be to enforce as many check on the smart contract side as possible

Recommendation

Check if `maxValue` is exceeded after swapping in `relayTokenIn` and `relayAndTransfer` functions.

[I-02] Users can be forced to receive gas in case of hanging allowance

Context

- GasbotV2.sol

Description

The current flow through the Gasbot UI is user either approving the GasbotV2 contract to spend their tokens or signing a permit message. In cases where the user has given higher allowance than the amount they want to swap, the permit message check is skipped but the allowance is deducted for only the amount they want to swap.

Both `relayTokenIn` and `relayAndTransfer` functions are callable only by the relayer, so the assumption is that the server side code checks if the person requesting the gas is indeed the person who signed the permit message.

A problem can arise if a user gives a high/unlimited allowance to the GasbotV2 contract but the server side code checks only if permit message is valid, but not that it is signed by the same user who is requesting the gas.

This is an informational issue since it addresses server side code that is not in the scope of this audit.

Recommendation

Implement all needed checks on the server side.

[I-03] Minimum added value not checked

Context

- GasbotV2.sol

Description

The Gasbot documentation states that for each swap there is an associated fee model, including a minimum fee of $0.30. The current implementation of `swapGas` function does not check if the amount of gas received is greater than the minimum fee. In cases where user swaps a small amount of tokens, the fee can be higher than the value provided. In this case the user wouldn't receive any gas on the destination chain.

As the gas for swapping varies from chain to chain, minimum fee enforcement requires significant changes to the code. As this is a new feature, it is marked as informational.

Recommendation

As an approximate solution at least add a check for the `addedValue` to be greater than $10.