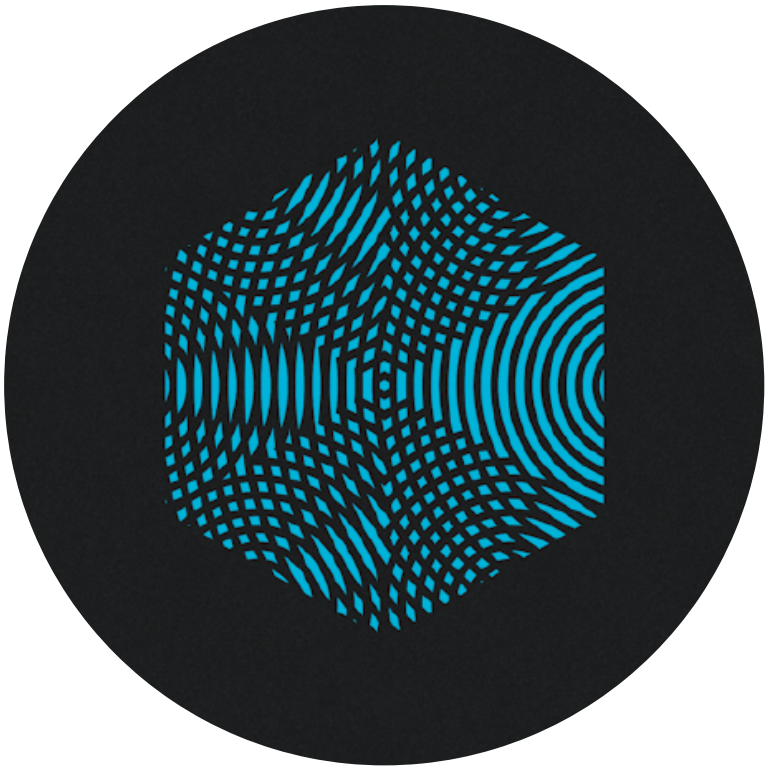

Gasbot v2 Security Review

Conducted by: Kristian Apostolov

January 7th, 2024

Contents

1. About Kristian Apostolov

Kristian Apostolov, also known as Chriss is an EVM smart contract security researcher. He is currently a Security Researcher at Guardian Audits and a security contest competitor.

You can reach out for security consulting on his Telegram, Twitter, or Cantina.

2. Disclaimer

Security reviews are a time, resource and expertise bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to find as many vulnerabilities as possible. Security reviews can show the presence of vulnerabilities **but not their absence**.

3. Introduction

A time-boxed security review of the **GasBot v2** was conducted by Kristian Apostolov, with a focus on the security aspects of the application's smart contract implementation.

4. About Gasbot v2

GasBot v2 serves as a cross-chain gas solution, simplifying the process of obtaining gas on any EVM chain for users. Version 2 includes a gas swap feature, allowing users to trade native tokens from one chain for those on another chain.

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* - efb5e1d3735f24c7fadb17d59247a262e2647c7b**

***fixes review commit hash* -**

7. Executive Summary

Over the course of the security review, Kristian Apostolov engaged with Gasbot.xyz to review Gasbot v2. In this period of time a total of **10** issues were uncovered.

Protocol Summary

Protocol Name	Gasbot v2
Repository	GasBot-xyz
Date	January 7th, 2024
Protocol Type	Multichain Gas Service
SLOC	239

Findings Count

Severity	**Amount**
Medium	1
Low	4
QA	5
Total Findings	**10**

Summary of Findings

ID	Title	Severity	Status
[M-01]	Polygon re-orgs can cause the protocol to lose funds	Medium	Unresolved
[L-01]	block.timestamp used as deadline for Uniswap swaps	Low	Unresolved
[L-02]	_toChainId being a uint16 dissallows the protocol from supporting chains with IDs greater than 65535	Low	Unresolved
[L-03]	No minimum amount of tokens required in swapGas()	Low	Unresolved
[L-04]	Swap paths are restricted to using a single pool	Low	Unresolved
[QA-01]	Wrong NatSpec comments	QA	Unresolved
[QA-02]	Use guard clauses to reduce nesting	QA	Unresolved
[QA-03]	No event emission in transferGasOut()	QA	Unresolved
[QA-04]	Only last relayer's call gets given a gas limit in replenishRelayers()	QA	Unresolved
[QA-05]	Reverts without a message	QA	Unresolved

8. Findings

8.1. Medium Findings

[M-01] Polygon re-orgs can cause the protocol to lose funds

Severity

Impact: High, since it can lead to a loss of funds for the protocol and/or the user.

Likelihood: Low, since deep reorgs are not particularly common.

Description

The protocol transfers users' funds to another chain through a relayer. The relayer picks up a user request through the `GasSwap` event emitted by `swapGas()`. That transaction then gets executed on the destination chain as soon as possible. The issue arises if the origin/destination chain undergoes a reorganization(fork). A chain reorg can occur when multiple miners mine valid blocks simultaneously. In such a case, the network needs to decide which block to add or remove from the blockchain. Under such a scenario either the payment transaction on the origin chain, or the funds release on the destination chain will get wiped out and one of the sides will lose funds. Read more about reorgs and their effects here.

Recommendations

Consider waiting 45s to 1m before executing requests with a destination/origin chain that is know to undergo reorgs on a regular basis. Such chains are Polygon and BCS to a lesser extent.

8.2. Low Findings

[L-01] block.timestamp used as deadline for Uniswap swaps

Severity

Impact: Low, since this will pose an issue only in a case where a user's transaction stays in the mempool for a prolongued period of time.

Likelihood: High, due to flawed timestamps being passed on every uniswap swap.

Description

Using `block.timestamp` as an expiration timestamp for a uniswap swap sets the expiration to the block, in which the transaction gets mined in. This will cause transactions leveraging a swap, which stayed in the mempool for a long time, to get executed with a no-longer valid `_minAmountOut`.

Recommendations

Consider explicitly passing a timestamp to `_swap()` instead of using `block.timestamp`.

[L-02] `_toChainId` being a `uint16` dissallows the protocol from supporting chains with IDs greater than 65535

Severity

Impact: Very low. It is very unlikely with for such a chain to be integrated with.

Likelihood: Very low, since there are not a lot of chains with IDs greater than the 16-bit integer limit.

Description

The `_toChainId` parameter of `swapGas()` is a `uint16`, which means that the protocol cannot support chains with IDs greater than the max 16-bit `uint`. This will dissallow the protocol from supporting chains with IDs greater than 65535.

Recommendations

Consider changing the `_toChainId` parameter to a `uint256` to allow for chains with IDs greater than 65535 to be supported.

[L-03] No minimum amount of tokens required in `swapGas()`

Severity

Impact: Low, not much can be caused.

Likelihood: Low. Users can call `swapGas()` with a very small amount, though their request will not be fulfilled.

Description

`swapGas()` allows users to swap the native token on one chain for the same value's worth of native tokens on another chain. Users can pass whatever amount they would like to swap, that is bellow `maxValue`. The issue here is that there is no minimum users cannot pass bellow. This will cause requests with an amount lower than necessary for covering the gas costs of the swap will not be fulfilled, but will still be taken in by the protocol.

Recommendations

Consider creating a minimum amount limit to preven unwanted edge cases. Also consider clearly documenting this type of behaviour for the users.

[L-04] Swap paths are restricted to using a single pool

Severity

Impact: Medium, since this can affect the slippage on swaps.

Likelihood: Very low as most stable `coin -> native token` pairs are likely to have deep liquidity.

Description

The protocol uses Uniswap swaps to swap between stable coins and native tokens. The issue here is that there may not `coin -> native token` pairs that have enough liquidity to support the swap with appropriate slippage. This will result in performing a swap with a suboptimal slippage, resulting in a worse rate for the user.

Recommendations

Consider allowing multi-pool swaps by implementing some sort of a setter for intermediate pools.

8.3. QA Findings

[QA-01] Wrong NatSpec comments

Description

There are multiple instances where the NatSpec comments are not correct.

Recommendations

`_gasLimit` isn't in the NatSpec of `transferGasOut()`, `relayAndTransfer()`, and `replenishRelayers()`. Consider adding it.

`.transfer()` notice in the NatSpec `swapGas()` is not needed. Consider removing it or replacing it with a notice about using `.call()` with a gas limit.

`setRelayer()`'s `_status` parameter is not present in the function's NatSpec either. Consider adding it as well.

There are also multiple functions without NatSpec comments. Though most of them are not user facing, consider clearly documenting their behavior - `_permitAndTransferIn()`, `_swap()`, `_unwrap()`, `_transferAtLeast()`, `getRelayerBalances()`, `getDefaultSwapPaths`.

[QA-02] Use guard clauses to reduce nesting

Description

There are multiple intances where guard clauses can be used to reduce nesting and improve readability.

Recommendations

Consider changing the following code:

```
if (i == length - 1) {
    _transferAtLeast
    //(_relayers[i], _amounts[i], _gasLimit); // Any extra goes to the last relayer
} else {
    (bool success, ) = payable(_relayers[i]).call{
        value: _amounts[i]
    }("");
    require(success, "Transfer failed");
}
```

to:

```
if (i == length - 1) {
    _transferAtLeast
    //(_relayers[i], _amounts[i], _gasLimit); // Any extra goes to the last relayer
    return;
}
(bool success, ) = payable(_relayers[i]).call{
    value: _amounts[i]
}("");
require(success, "Transfer failed");
```

Consider changing the following code:

```
if (_recipient != address(0)) {
    (success, ) = payable(_recipient).call{
        value: address(this).balance
    }("");
    require(success, "Transfer failed");
}
```

to:

```
if (_recipient == address(0)) {
    return;
}
// ...
```

Consider changing the following code:

```
if (isV3Router) {
    if (_toWeth) {
        uniV3Path = abi.encodePacked(
            homeToken,
            defaultPoolFee,
            address(WETH)
        );
    } else {
        uniV3Path = abi.encodePacked(
            address(WETH),
            defaultPoolFee,
            homeToken
        );
    }
} else {
    uniV2Path = new address[](2);
    if (_toWeth) {
        uniV2Path[0] = homeToken;
        uniV2Path[1] = address(WETH);
    } else {
        uniV2Path[0] = address(WETH);
        uniV2Path[1] = homeToken;
    }
}
```

to:

```
if (isV3Router) {
    uniV3Path = abi.encodePacked(
        _toWeth ? homeToken : address(WETH),
        defaultPoolFee,
        _toWeth ? address(WETH) : homeToken
    );
} else {
    uniV2Path = new address[](2);
    uniV2Path[0] = _toWeth ? homeToken : address(WETH);
    uniV2Path[1] = _toWeth ? address(WETH) : homeToken;
}
```

Consider changing the following code:

```
if (_uniV3Path.length > 0) {
    IUniswapRouterV3(_router).exactInput(
        IUniswapRouterV3.ExactInputParams({
            path: _uniV3Path,
            recipient: address(this),
            deadline: block.timestamp,
            amountIn: _amount,
            amountOutMinimum: _minAmountOut
        })
    );
} else {
    IUniswapRouterV2(_router).swapExactTokensForTokens(
        _amount,
        _minAmountOut,
        _uniV2Path,
        address(this),
        block.timestamp
    );
}
```

to:

```
if (_uniV3Path.length > 0) {
    IUniswapRouterV3(_router).exactInput(
        IUniswapRouterV3.ExactInputParams({
            path: _uniV3Path,
            recipient: address(this),
            deadline: block.timestamp,
            amountIn: _amount,
            amountOutMinimum: _minAmountOut
        })
    );
    return;
}
IUniswapRouterV2(_router).swapExactTokensForTokens(
    _amount,
    _minAmountOut,
    _uniV2Path,
    address(this),
    block.timestamp
);
```

[QA-03] No event emission in `transferGasOut()`

Description

No event gets emitted when transferring the native token out to the user.

Recommendations

Consider emitting an event in `transferGasOut()` after successfully transferring the funds to the user.

[QA-04] Only last relayer's call gets given a gas limit in `replenishRelayers()`

Description

`replenishRelayers()` sents gas to relayers in order for them to be able to relay protocol requests. All `address.call`s in the protocol follow a pattern of passing a gas allowance for the request, though the calls in `replenishRelayers()` do not.

```
// _transferAtLeast()
(bool success, ) = payable(_recipient).call{
    value: address(this).balance,
    gas: _gasLimit
}("");
```

```
// replenishRelayers()
else {
    (bool success, ) = payable(_relayers[i]).call{
        value: _amounts[i]
    }("");
    require(success, "Transfer failed");
}
```

Recommendations

Consider adding a gas allowance to the `address.call` in `replenishRelayers()` in order to be consistent with the rest of the logic.

[QA-05] Reverts without a message

Description

There are multiple instances where the protocol reverts without an error message.

Recommendations

Consider adding error messages to the following reverts:

```
54: require(_owner != address(0));
55: require(_defaultRouter != address(0));
56: require(_weth != address(0));
57: require(_homeToken != address(0));
```

```
312: require(_defaultRouter != address(0));
```

```
320: require(_defaultPoolFee > 0);
```

```
329: require(_homeToken != address(0));
```

```
367: require(success);
```