BURRA SEC

ProphetRouter Security Review

Reviewed by: windhustler

ProphetRouter Security Review Report

Burra Security

Feb 7, 2024

Introduction

A time-boxed security review of the **ProphetRouter** 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 inter-operability protocols and their integrations.

About ProphetRouter

ProphetRouterV1 is a router contract built to route all trade done through the bot and take a revshare fee that is withdrawable by the team.

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* - 4c91c1469eb6c64280ae6ceb79bb5221b92fbccb**

Scope

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

- `contracts/ProphetRouterV1.sol`

Findings Summary

ID	Title	Severity	Status
[H-01]	`ProphetMaxBuy` always reverts due to `amountOutMin` not being adjusted during iteration	High	Fixed

ID	Title	Severity	Status
[M-01]	`ProphetMaxBuy` function is not gas efficient and might not be able to find exact value if the passed `msg.value` is too high	Medium	Ack
[M-02]	`ProphetSmartSell` function doesn't yield the exact output amount as you would expect	Medium	Ack
[M-03]	`swapTokensSupportingFeeOnTransferTokensForExactETH` can leave output tokens hanging in the `ProphetRouterV1` contract	Medium	Fixed
[L-01]	Improvements to fee mechanism in `swapTokensSupportingFeeOnTransferTokensForExactETH`	Low	Ack

Detailed Findings

[H-01] - `ProphetMaxBuy` always reverts due to amountOutMin not being adjusted during iteration

Context

- ProphetRouterV1.sol

Description

`amountOutMin` is used to protect the user against excessive slippage during the execution of the swap. As the function `ProphetMaxBuy` tries decrementing the `amountIn` to account for tokens that have a max transfer limit per transaction or pools with low liquidity, it doesn't adjust the `amountOutMin` accordingly.

The expected user behaviour is to pass the `amountOutMin` based on the `msg.value` passed in the function. If the execution enters the `if (tokenToEther[tokenAddress] == 0){` branch and amountIn gets decremented by 10%, the `amountOutMin` will get outdated and the function will revert.

Recommendation

Add the following line after the `amountIn` decrement:

```
} catch {
    amountIn = (amountIn * 9_000) / MAX_BIPS;
    amountOutMin = (amountOutMin * 9_000) / MAX_BIPS;
    continue;
}
```

Consider this change together with the other recommendations in this report.

Resolution

Fixed.

[M-01] - `ProphetMaxBuy` function is not gas efficient and might not be able to find exact value if the passed `msg.value` is too high

Context

- ProphetRouterV1.sol

Description

`ProphetMaxBuy` has logic to find the maximum amount of tokens that can be traded. It iterates through 10 steps at each step decrementing the `amountIn` by 10% and checking if the `ProphetBuy` function reverts.

Let's take a simple example of a token having max transfer limit of 100 tokens while the user passes `msg.value` of 500 tokens. At the final iteration the `amountIn` will be 500 * 0.9^10 = 174.33 tokens which will still be higher than the max transfer limit of 100 tokens and the function will do nothing but spend a lot of gas for the user.

Recommendation

The logic for finding the maximum `amountIn` shouldn't be part of the `ProphetRouterV1.sol` contract at all, and such a search should be performed off-chain. A more efficient way of finding the `amountIn` value should be performed by binary search, which will arrive at a much more precise value.

Resolution

This is intended to operate this way, the team is aware of the gas inefficiencies and we have implemented some methods to avoid having users pay high gas fees such as storing the `maxBuyAmount` for each token address.

[M-02] - `ProphetSmartSell` function doesn't yield the exact output amount as you would expect

Context

- ProphetRouterV1.sol

Description

The intention of `ProphetSmartSell` function is to enable the user to specify the exact amount of ETH he wants to receive in exchange for some ERC20 token. If we examine the parameters of the function, `amountOut` should be the exact amount of ETH while `amountInMax` should be the maximum amount of ERC20 tokens that the user is willing to spend.

To understand why the function doesn't work as expected with fee on transfer tokens it is useful to look at `UniswapV2Router01.sol` and `UniswapV2Router02.sol` from https://github.com/Uniswap/v2-periphery.

If we observe the difference in the implementation of various swap functions with regular tokens versus fee on transfer tokens, we can see:

- fee on transfer tokens cannot rely on the `UniswapV2Library` to calculate the amounts prior to the actual transfer.
- But instead the input token amount is first transferred to the `UniswapV2Pair` contract and then based on the actual transferred amount the output amount is calculated.
- This is necessary as different tokens have different transfer fees so actual transferred amounts can vary a lot.

As a consequence there are no `exactOutput` functions for fee on transfer tokens, only `exactInput`, e.g. `swapExactTokensForTokensSupportingFeeOnTransferTokens`, `swapExactETHForTokensSupportingFeeOnTransferTokens`.

Coming back to the `ProphetSmartSell` function:

```
uint[] memory amounts = UniswapV2Library.getAmountsIn(factory,
    amountOut + feeAmount, path);
```

The `amount[0]` quantity is pre-transfer and is not accurate, i.e. by providing `amount[0]` quantity of tokens the swap will not generate `amountOut + feeAmount` amount of ETH.

As a consequence the user will always get less than the amountOut he specified in the function.

Recommendation

It's hard to recommend a specific fix for this function as it is not clear what the intention of the function is. If the intention is to receieve the exact amount of ETH, the function fails to do so and should be removed.

Resolution

This function works as intended as per the tech specs. Acknowledged.

[M-03] - `swapTokensSupportingFeeOnTransferTokensForExactETH` can leave output tokens hanging in the ProphetRouterV1 contract

Context

- ProphetRouterV1.sol

Description

`swapTokensSupportingFeeOnTransferTokensForExactETH` function should be identical to the `swapExactTokensForETHSupportingFeeOnTransferTokens` from `UniswapV2Router02.sol` except for the fee inclusion. By observing the `UniswapV2Router02.sol` code, we can see that the first line asserts that the last token in the path is WETH, `require(path[path.length - 1] == WETH, 'UniswapV2Router: INVALID_PATH');`. This is not the case in the `ProphetRouterV1.sol` contract, which can lead to the output tokens being stuck in the contract as only WETH is transferred out.

Another observation is that there is `withdraw` function to withdraw any tokens from the contract. This would indicate the intention to maybe have the contract hold some tokens. Although I couldn't find any case where the contract would hold `WETH` in normal circumstances, but if it would hold it an attacker could drain all the `WETH` from the contract by using the `swapTokensSupportingFeeOnTransferTokensForExactETH` function. This is possible as all the WETH is withdrawn from the contract since it is not checked if the output token from the swap is actually WETH.

Recommendation

Add the following line at the beginning of the `swapTokensSupportingFeeOnTransferTokensForExactETH` function:

```
require(path[path.length - 1] == WETH, 'UniswapV2Router: INVALID_PATH')
    ;
```

Also transfer all the ETH generated from fees to the `owner` not leaving it hanging in the contract. `ProphetRouterV1.sol` should not be holding any tokens.

Resolution

Fixed.

[L-01] - Improvements to fee mechanism in `swapTokensSupportingFeeOnTransferTokensForExactETH`

Context

- ProphetRouterV1.sol

Description

The last line in the `swapTokensSupportingFeeOnTransferTokensForExactETH` function decrements the passed fee from the amount of WETH out. As the output amount is not known beforehand it can occur that the fee amount is overestimated and the function reverts.

Recommendation

Consider subtracting fee only if it is lower than the amount of WETH out, or passing it as a percentage of the amount of WETH out.

Resolution

Acknowledged.

Informational findings

[I-01] - swapTokensSupportingFeeOnTransferTokensForExactETH name is misleading

`swapTokensSupportingFeeOnTransferTokensForExactETH` function is actually swapping an exact amount of ERC20 tokens for ETH.

It should rather be called `swapExactTokensForETHSupportingFeeOnTransferTokens`.

[I-02] - Save the balance of WETH after the swap in swapTokensSupportingFeeOnTransferTokensForExactETH to save gas

Balance of WETH after the swap is read twice in the `swapTokensSupportingFeeOnTransferTokensForExact` function: `solidity require( IERC20(path[path.length - 1]).balanceOf( address(this)).sub(balanceOfWETHBefore)>= amountOutMin, 'PropherRouter : INSUFFICIENT_OUTPUT_AMOUNT'); uint balanceOfWETHAfter = IERC20(path [path.length - 1]).balanceOf(address(this));`

Change this to:

```
uint balanceOfWETHAfter = IERC20(path[path.length - 1]).balanceOf(
    address(this));
 require(
    balanceOfWETHAfter.sub(balanceOfWETHBefore) >= amountOutMin,
    'PropherRouter: INSUFFICIENT_OUTPUT_AMOUNT'
);
```

[I-04] - `ProphetBuy` and `ProphetMaxBuy` unnecessary checks if the hardcoded `WETH` address is the first token in the path

```
address[] memory path = getPathForTokenToToken(true, tokenAddress);
```

This function already returns WETH as the first token in the path so the following check is unnecessary:

```
require(path[0] == WETH, 'PropherRouter: INVALID_PATH');
```

[I-05] - `ProphetSell` and `ProphetSmartSell` unnecessary checks if the hardcoded `WETH` address is the last token in the path

```
address[] memory path = getPathForTokenToToken(false, tokenAddress);
```

This function already returns WETH as the last token in the path so the following check is unnecessary:

```
require(path[path.length - 1] == WETH, 'PropherRouter: INVALID_PATH');
```

[I-06] - Redundant break in `ProphetMaxBuy` function

If try `this.ProphetBuy...` block is successful `isSwapComplete`is set to true and the while loop is exited. `break;` is therefor redundant.

Resolution

Fixed.