Zellic

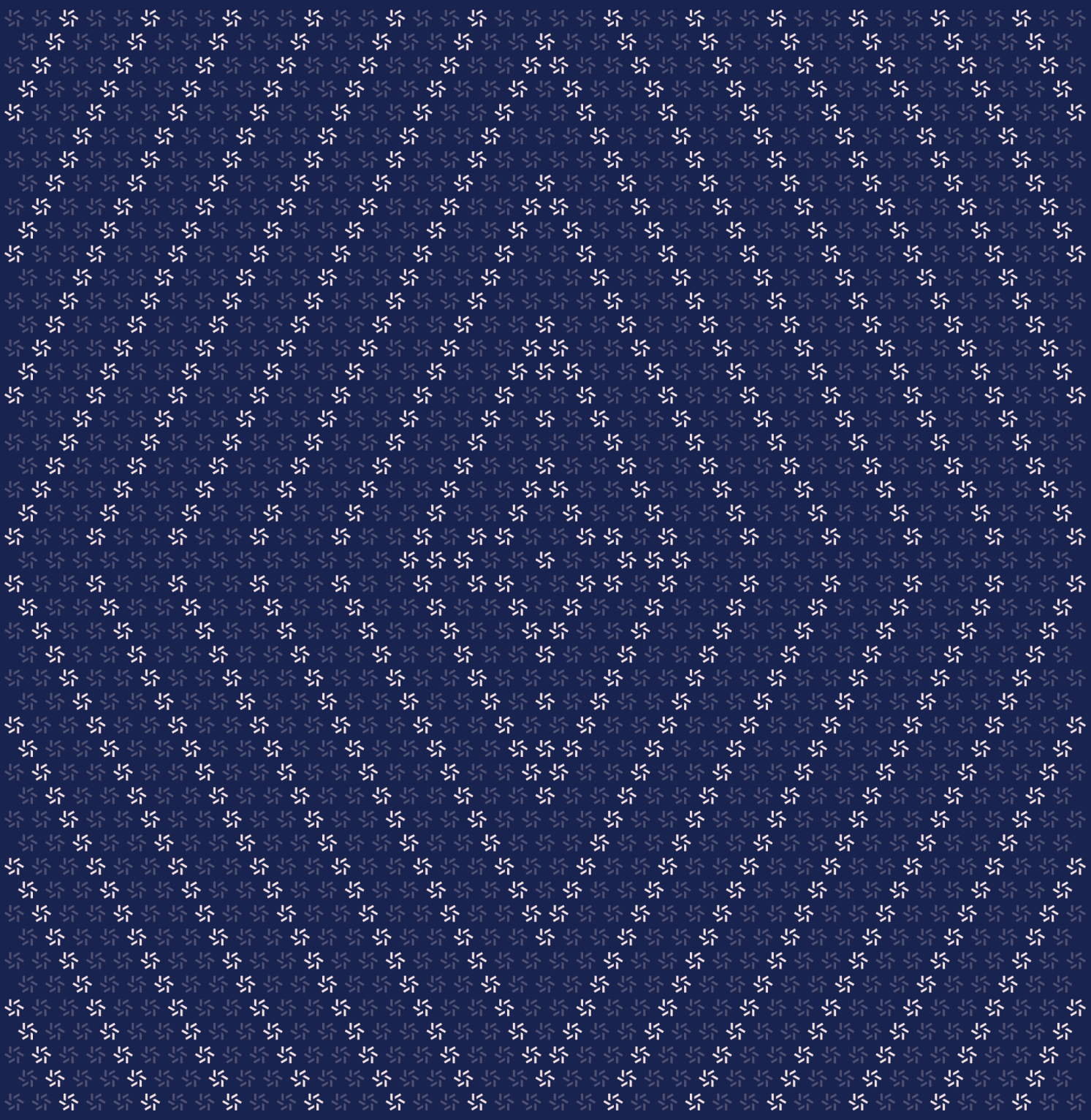
Prepared for
Bruno Wu
Ooga Booga

Prepared by
Junghoon Cho
Vlad Toie
Zellic

September 26, 2024

Ooga Booga

Smart Contract Security Assessment



Contents

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for Ooga Booga from September 18th to September 20th, 2024. During this engagement, Zellic reviewed Ooga Booga's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Does the router contract let users safely swap their tokens?
- Are there issues in protocol math or logic that lead to loss of funds?
- Does the project have any centralization risks?
- Is Ooga Booga secure against common smart contract vulnerabilities?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- OBExecutor contract
- Front-end components
- Infrastructure relating to the project
- Key custody

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

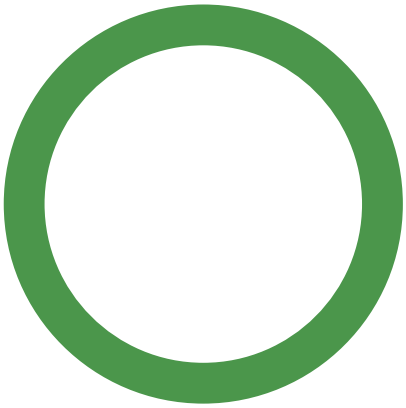
1.4. Results

During our assessment on the scoped Ooga Booga contracts, we discovered two findings, both of which were low impact.

Additionally, Zellic recorded its notes and observations from the assessment for the benefit of Ooga Booga in the Discussion section (4. ↗).

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	0
Low	2
Informational	0



2. Introduction

2.1. About Ooga Booga

Ooga Booga contributed the following description of Ooga Booga:

> Ooga Booga is Berachain's native liquidity aggregator.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and

Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped contracts itself. These observations — found in the Discussion (4. ↗) section of the document — may include suggestions for improving the codebase, or general recommendations, but do not necessarily convey that we suggest a code change.

2.3. Scope

The engagement involved a review of the following targets:

Ooga Booga Contracts

Type	Solidity
Platform	EVM-compatible
Target	oogabooga-router
Repository	https://github.com/0xoogabooga/oogabooga-router ↗
Version	`c03197bff1f06a2962e019e3d0bb7351142dd97d`

Programs	`OBRouter.sol`
	`TokenHelper.sol`
	`OnlyApproved.sol`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of 1.5 person-days. The assessment was conducted by two consultants over the course of three calendar days.

Contact Information

The following project managers were associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io

Chad McDonald
Engagement Manager
chad@zellic.io

The following consultants were engaged to conduct the assessment:

Junghoon Cho
Engineer
junghoon@zellic.io

Vlad Toie
Engineer
vlad@zellic.io

2.5. Project Timeline

The key dates of the engagement are detailed below.

September 18, 2024	Kick-off call
September 18, 2024	Start of primary review period
September 20, 2024	End of primary review period

3. Detailed Findings

3.1. The `tokenInfo.outputQuote` can be cast to a negative number

Target	OBRouter.sol		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	Low	**Impact**	Low

Description

The `_swap` function calculates the slippage that occurs during the token swap and uses it to determine the amount of tokens (`amountOut`) the user will receive. Slippage is calculated by subtracting the user-defined quote (`tokenInfo.outputQuote`) from the fee-adjusted amount of tokens the user will receive. In the case of positive slippage, `amountOut` is set to `tokenInfo.outputQuote`.

```
int256 slippage = int256(amountOut) - int256(tokenInfo.outputQuote);
if (slippage > 0) {
    amountOut = tokenInfo.outputQuote;
}
```

To calculate the slippage from the unsigned integers `amountOut` and `tokenInfo.outputQuote`, both variables are cast to `int256`. However, if the value of `tokenInfo.outputQuote` exceeds the maximum value for `int256`, it will be converted to a negative number.

Impact

If `int256(tokenInfo.outputQuote)` is converted to a negative value, the `slippage` becomes a positive value, causing `amountOut` to be set to a very large value (`2 ** 255` or above). Eventually, the contract will attempt to transfer an excessively large amount of tokens to the address specified by the user. However, considering the general total supply of tokens, it is highly unlikely that such a transfer would succeed and result in the contract being drained.

Recommendations

Add a check to ensure that `tokenInfo.outputQuote` does not exceed `2 ** 255`, or use a library such as SafeCast.

Remediation

This issue has been acknowledged by Ooga Booga, and a fix was implemented in commit `aba9c32e` ↗.

3.2. User native funds can be locked in the contract

Target	OBRouter.sol		
Category	Coding Mistakes	**Severity**	Low
Likelihood	N/A	**Impact**	Low

Description

The `_swapApproval` function handles the receival of ERC-20 and native tokens to be used in the swap operation.

In the case that the input token is native, the function checks whether `msg.value == tokenInfo.inputAmount`, which essentially ensures that the correct amount of native tokens are sent to the contract. In the else case, the function receives ERC-20 tokens, but it does not check whether no native tokens have been provided by mistake (i.e., `msg.value > 0`).

```
function _swapApproval(
    // ...
) internal returns (uint256 amountOut) {

    if (tokenInfo.inputToken == TokenHelper.NATIVE_TOKEN) {
        // Support rebasing tokens by allowing the user to trade the entire
    balance
        if (tokenInfo.inputAmount == 0) {
            tokenInfo.inputAmount = msg.value;
        } else {
            require(
                msg.value == tokenInfo.inputAmount,
                NativeDepositValueMismatch(tokenInfo.inputAmount, msg.value)
            );
        }
    } else {
        // Support rebasing tokens by allowing the user to trade the entire
    balance
        if (tokenInfo.inputAmount == 0) {
            tokenInfo.inputAmount
    = IERC20(tokenInfo.inputToken).balanceOf(msg.sender);
        }
        IERC20(tokenInfo.inputToken).safeTransferFrom(msg.sender, executor,
    tokenInfo.inputAmount);
    }
    // ...
```

```
}
```

Impact

This leads to a situation where the user might mistakenly send and lock native tokens in the contract.

Recommendations

We recommend adding a check to ensure that no native tokens are sent to the contract when the input token is an ERC-20 token:

```
if (tokenInfo.inputToken == TokenHelper.NATIVE_TOKEN) {
    // ...
} else {
   require(
       msg.value == 0,
       NativeNotAllowed(tokenInfo.inputToken, msg.value)
   );
    // ...
}
```

Remediation

This issue has been acknowledged by Ooga Booga, and a fix was implemented in commit 2ae6802c ↗.

4. Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security related and do not convey that we are suggesting a code change.

4.1. Suggested improvements for variable and function naming

These are some suggested improvements for variable and function naming.

Function `unpaused`

The following function exists to unpause the OBRouter contract that has been paused by the `pause` function.

```
/// @dev Unpause swap restricted to the owner
function unpaused() external onlyOwner {
    _unpause();
}
```

Given the purpose of the function, consider renaming it to `unpause`.

Variable `approvedAddresses`

The following variable is a whitelist that designates the addresses allowed to transfer funds held in the OBRouter contract using the `transferRouterFunds` function.

```
mapping(address => bool) private approvedAddresses;
```

Given that this variable maps an address to a boolean, consider renaming it to `approved`.

4.2. Suggested `tokenInfo` constraints

The `tokenInfo` struct is used to store information about the input and output tokens of the swap operation. It is defined as follows:

```
struct swapTokenInfo
{
    address inputToken;
    uint256 inputAmount;
```

```
    address outputToken;
    uint256 outputQuote;
    uint256 outputMin;
    address outputReceiver;
}
```

We note that the swap function utilizes a parameter called `pathDefinition`, which defines the path for the swap operation. This parameter is not checked against the `tokenInfo` struct, which could lead to severe inconsistencies. For example, a user could use a malicious `inputToken` with a path that defines a legitimate token exchange, stealing funds from the OBRouter.

As of right now, however, there are no direct security implications, as the OBRouter does not hold funds by itself. For posterity, we recommend defining explicit constraints for the `tokenInfo` struct to ensure that the `pathDefinition` parameter is consistent with the `tokenInfo` struct. One such constraint could be ensuring that the `inputToken` and `outputToken` fields of the `tokenInfo` struct are part of the `pathDefinition` parameter, at the right indexes.

5. Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the contracts and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

5.1. Module: OBRouter.sol

Function: `swapPermit2(permit2Info permit2, swapTokenInfo tokenInfo, bytes pathDefinition, address executor, uint32 referralCode)`

This function allows withdrawing funds via the Permit2 standard.

Inputs

- `permit2`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None — assumed to contain a valid Permit2 signature.
  - **Impact**: The `permit2` object is used to validate the transfer of funds.
- `tokenInfo`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None — assumed to contain valid token information that would check out with the Permit2 signature.
  - **Impact**: The token information is used to determine the input and output tokens, the input amount, the output quote, the output minimum, and the output receiver.
- `pathDefinition`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None — assumed to be a valid path definition.
  - **Impact**: The path of the swap.
- `executor`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None — assumed to be a valid executor.
  - **Impact**: The executor of the swap.
- `referralCode`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None — assumed to be a valid referral code.
  - **Impact**: The referral code of who to forward the fees to.

Branches and code coverage

Intended branches

- Construct the `PermitTransferFrom` object from the `tokenInfo` and `permit2` objects.
  - ☑ Test coverage
- Construct the `SignatureTransferDetails` object from the `executor` and `tokenInfo` objects.
  - ☑ Test coverage
- Call `permitTransferFrom`, which should validate the signature and transfer the funds.
  - ☑ Test coverage

Negative behavior

- Should not allow anyone other than `msg.sender` to be the owner of the funds in the `permitTransferFrom` function. Otherwise, the swap can be front-run.
  - ☑ Negative test

Function: `transferRouterFunds(address[] tokens, uint256[] amounts, address dest)`

This function allows the owner to transfer funds held by the router contract.

Inputs

- `tokens`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Array length must be equal to `amounts`.
  - **Impact**: The tokens to be transferred.
- `amounts`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Array length must be equal to `tokens`.
  - **Impact**: The amounts of each token to be transferred.
- `dest`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: The address to which the funds should be sent.

Branches and code coverage

Intended branches

- Loop over the tokens and transfer the amounts to the destination.
  - ☑ Test coverage

- If the token is the native token, deposit the amount to WETH and transfer it.
  - ☑ Test coverage

Negative behavior

- Should not be called with malicious intent. Technically, this should never be the case as funds are never purposely stored in the contract.
  - ☑ Negative test

Function: `_swapApproval(swapTokenInfo tokenInfo, bytes pathDefinition, address executor, uint32 referralCode)`

This function validates input/output tokens. It also draws the input tokens from the user and sends them to the executor.

Inputs

- `tokenInfo`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Checks that the input token is not the same as the output token and that the amount of tokens is the same as `msg.value`. Anything else is controlled by the caller.
  - **Impact**: The `tokenInfo` object is used to determine the input and output tokens, the input amount, the output quote, the output minimum, and the output receiver.
- `pathDefinition`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None. Assumed to be valid — it will eventually be checked by the third-party contract that executes the swap.
  - **Impact**: The path of the swap.
- `executor`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None.
  - **Impact**: The executor of the swap.
- `referralCode`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None at this level — checked in `_swap`.
  - **Impact**: The referral code of who to forward the fees to.

Branches and code coverage

Intended branches

- If the input token is the native token, ensure that `msg.value` is the same as the input amount.
  - ☑ Test coverage
- If the input token is not the native token, transfer the input amount from the user to the executor as ERC-20 tokens, denominated in the input token.
  - ☑ Test coverage
- If the `inputAmount` is set to `0` and the input token is the native token, set the input amount to `msg.value`.
  - ☑ Test coverage
- If the `inputAmount` is set to `0` and the input token is not the native token, set the input amount to the balance of the input token in the user's wallet.
  - ☑ Test coverage

Negative behavior

- The `inputToken` should be validated or whitelisted. A malicious token can be used with a legitimate path to drain the contract. Currently, however, no funds are supposed to be in the contract, so this is not a concern.
  - ☐ Negative test
- Should not allow inputting a `tokenInfo.inputAmount` greater than `msg.value` if the input token is the native token. Ensured by the `require` statement.
  - ☑ Negative test

Function: `_swap(swapTokenInfo tokenInfo, bytes pathDefinition, address executor, uint32 referralCode)`

This is the pre- and post-swap logic sanity checks.

Inputs

- `tokenInfo`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None — assumed to contain valid token information that would check out with the Permit2 signature.
  - **Impact**: The token information is used to determine the input and output tokens, the input amount, the output quote, the output minimum, and the output receiver.
- `pathDefinition`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None — assumed to be a valid path definition.
  - **Impact**: The path of the swap.
- `executor`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None — assumed to be a valid executor.

    - **Impact**: The executor of the swap.
- `referralCode`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: None — assumed to be a valid referral code.
    - **Impact**: The referral code of who to forward the fees to.

Branches and code coverage

Intended branches

- Assume the executor has already received the input tokens.
    - ☑ Test coverage
- Call the `executePath` function of the executor contract with `pathDefinition` and `msg.value`.
    - ☑ Test coverage
- Calculate the `amountOut` by subtracting the balance before the swap from the balance after the swap.
    - ☑ Test coverage
- If the `referralCode` is greater than the `REFERRAL_WITH_FEE_THRESHOLD`, check if the beneficiary is not the contract itself and transfer the referral fee to the beneficiary.
    - ☑ Test coverage
- Calculate the fees and keep the remaining amount in the contract.
    - ☑ Test coverage
- Transfer the output tokens to the output receiver.
    - ☑ Test coverage

Negative behavior

- Should not allow the output minimum to be greater than the output quote.
    - ☑ Negative test
- Should not allow the output minimum to be zero.
    - ☑ Negative test
- Should not allow the input token to be the same as the output token.
    - ☑ Negative test
- Should not allow the output after fees to be less than the output minimum.
    - ☑ Negative test

6. Assessment Results

At the time of our assessment, the reviewed code was deployed to the Bartio Testnet.

During our assessment on the scoped Ooga Booga contracts, we discovered two findings, both of which were low impact.

6.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.