Zellic

DfynRFQ

Smart Contract Security Assessment

August 1, 2022

Prepared for:

Ramani Ramachandran and Priyeshu Garg

Router Protocol

Prepared by:

Katerina Belotskaia and Vlad Toie

Zellic Inc.

Contents

About Zellic

Zellic was founded in 2020 by a team of blockchain specialists with more than a decade of combined industry experience. We are leading experts in smart contracts and Web3 development, cryptography, web security, and reverse engineering. Before Zellic, we founded perfect blue, the top competitive hacking team in the world. Since then, our team has won countless cybersecurity contests and blockchain security events.

Zellic aims to treat clients on a case-by-case basis and to consider their individual, unique concerns and business needs. Our goal is to see the long-term success of our partners rather than simply provide a list of present security issues. Similarly, we strive to adapt to our partners' timelines and to be as available as possible. To keep up with our latest endeavors and research, check out our website zellic.io or follow @zellic_io on Twitter. If you are interested in partnering with Zellic, please email us at hello@zellic.io or contact us on Telegram at https://t.me/zellic_io.

1 Executive Summary

Zellic conducted an audit for Router Protocol from July 18th to July 20th, 2022.

Our general overview of the code is that it was well-organized and structured. The code coverage is adequate, and tests are included for the majority of the functions; however, it can be further extended and improved on. The documentation was minimal, and it could be improved. The code was easy to comprehend, and in most cases, intuitive.

We applaud Router Protocol for their attention to detail and diligence in maintaining high code quality standards in the development of DfynRFQ.

Zellic thoroughly reviewed the DfynRFQ codebase to find protocol-breaking bugs as defined by the documentation and to find any technical issues outlined in the Methodology section (2.2) of this document.

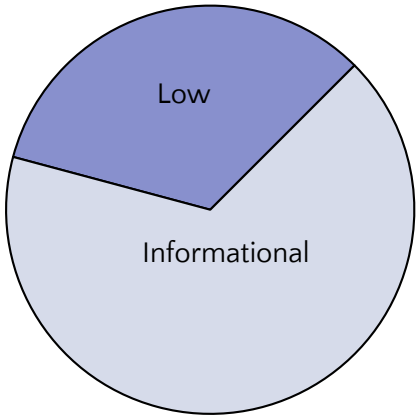
Specifically, taking into account DfynRFQ's threat model, we focused heavily on issues that would break core invariants such as the swaps signatures and their verification as well as how the fees are calculated and distributed. Moreover we thoroughly investigated the correctness of the swap functionality, such that it serves its intended purpose.

During our assessment on the scoped DfynRFQ contracts, we discovered 3 findings. Fortunately, no critical issues were found. Of the 3 findings, 1 was of low severity, and the remaining findings were informational in nature.

Additionally, Zellic recorded its notes and observations from the audit for Router Protocol's benefit in the Discussion section (4) at the end of the document.

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	0
Low	1
Informational	2



2 Introduction

2.1 About DfynRFQ

DfynRFQ is a multichain AMM DEX that is built to be an interconnected AMM with nodes spread on different blockchains, and those AMMs will be able to share liquidity and enable cross-chain swaps. Currently, Dfyn is live on Polygon and Fantom.

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 open-source 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. We analyze the scoped smart contract code using automated tools to quickly sieve out and catch these shallow bugs. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so forth 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 manually review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents. We also thoroughly examine the specifications and designs themselves for inconsistencies, flaws, and vulnerabilities. This involves use cases that open the opportunity for abuse, such as flawed tokenomics or share pricing, arbitrage opportunities, and so forth.

Complex integration risks. Several high-profile 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. We perform a meticulous review of all of the contract's possible external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so forth.

Code maturity. We review for possible 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, upgradeability weaknesses, centralization risks, and so forth.

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; we assign it on a case-by-case basis based on our professional judgment and experience. As one would expect, both the severity and likelihood of an issue affect its impact; for instance, a highly severe issue's impact may be attenuated by a very low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Similarly, Zellic organizes its reports such that the most important findings come first in the document rather than being 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 numerous soft factors, such as our clients' threat models, their business needs, their project timelines, and so forth. We aim to provide useful and actionable advice to our partners that consider their long-term goals rather than simply provide a list of security issues at present.

2.3 Scope

The engagement involved a review of the following targets:

DfynRFQ Contracts

Repository	https://github.com/dfyn/dfyn-RFQ/
Versions	`0fa84f86fa70046e572570c66c4f1bcb88c9897b`
Programs	• DfynRFQ
Type	Solidity
Platform	EVM-compatible

2.4 Project Overview

Zellic was contracted to perform a security assessment with two consultants for a total of two person-days.

Contact Information

The following project managers were associated with the engagement:

Jasraj Bedi, Co-founder
jazzy@zellic.io

Stephen Tong, Co-founder
stephen@zellic.io

The following consultants were engaged to conduct the assessment:

Katerina Belotskaia, Engineer
kate@zellic.io

Vlad Toie, Engineer
vlad@zellic.io

2.5 Project Timeline

The key dates of the engagement are detailed below.

July 18, 2022	Kick-off call
July 18, 2022	Start of primary review period
July 20, 2022	End of primary review period

3 Detailed Findings

3.1 Same token swap is allowed

- **Target**: DfynRFQ
- **Category**: Business Logic
- **Likelihood**: Medium
- **Severity**: Low
- **Impact**: Low

Description

A user might mistakenly perform a same-token swap via the protocol, since there are no restrictions against that.

Impact

In function `_swap()` there are no checks whatsoever for whether the `tokens[0]` and `tokens[1]` are identical.

```
function _swap(
    address custodian,
    address[] calldata tokens,
    uint256[] calldata amounts,
    uint64 deadline,
    bytes calldata signature
) internal onlyWhitelisted(custodian) returns (bool) {
    Swap memory swap = Swap({
        user: msg.sender,
        custodian: custodian,
        token0: tokens[0],
        token1: tokens[1],
        amount0: amounts[0],
        amount1: amounts[1],
        deadline: deadline,
        nonce: nonces[msg.sender],
        chainId: chainId
    });

    require(block.timestamp < swap.deadline, “Expired Order”);
    require(verify(swap, signature), “Invalid Signer”);
    require(swap.amount1 > 0 && swap.amount0 > 0, “amount ≠ 0”);
```

This can lead to loss of the gas cost used in the transaction, as well as the tokens lost to protocol fees, all due to an undesireable action performed by the user in the first place.

Recommendations

We recommend adding an additional check when performing a swap, such that the tokens on either side of the **swap** are not the same.

```
function _swap(
    address custodian,
    address[] calldata tokens,
    uint256[] calldata amounts,
    uint64 deadline,
    bytes calldata signature
) internal onlyWhitelisted(custodian) returns (bool) {
    require(tokens[0] ≠ tokens[1], “Same token swap is disallowed”);
    Swap memory swap = Swap({
        user: msg.sender,
        custodian: custodian,
        token0: tokens[0],
        token1: tokens[1],
        amount0: amounts[0],
        amount1: amounts[1],
        deadline: deadline,
        nonce: nonces[msg.sender],
        chainId: chainId
    });

    require(block.timestamp < swap.deadline, “Expired Order”);
    require(verify(swap, signature), “Invalid Signer”);
    require(swap.amount1 > 0 && swap.amount0 > 0, “amount ≠ 0”);
```

Remediation

This issue has been acknowledged by the Router team and mitigated in commit 3be1183.

3.2 DfynRFQ provides a function to renounce ownership

- **Target**: DfynRFQ
- **Category**: Business Logic
- **Likelihood**: N/A
- **Severity**: Informational
- **Impact**: Informational

Description

The DfynRFQ contract implements Ownable functionality, which provides a method named renounceOwnership that removes the current owner. This is likely not a desired feature.

Impact

If `renounceOwnership` were called, the contract would be left without an owner.

Recommendations

Override the `renounceOwnership` function:

```
function renounceOwnership() public override onlyOwner{
    revert("This feature is not available.");
}
```

Remediation

This issue has been mitigated by the Router team in commit 3be1183.

3.3 `msg.sender.transfer()` function usage

- **Target**: DfynRFQ
- **Category**: Business Logic
- **Likelihood**: N/A
- **Severity**: Informational
- **Impact**: Informational

Description

The `swapTokenToNative` calls the `transfer` function to send requested ether amount to `msg.sender` account.

Impact

The `transfer` function uses a hardcoded amount of GAS and will fail if GAS costs increase in the future.

Recommendations

Consider using `msg.sender.call.value(value)("")` function:

```
(bool success, ) = msg.sender.call.value(amounts[1].sub(feeAmount))("");
require(success, "Transfer failed.");
```

Remediation

This issue has been mitigated by the Router team in commit 3be1183.

4 Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment.

4.1 Array indexes may be out of bounds

In the case of the `tokens` and `amounts` array used in the `_swap()` function, no check on their length is performed. There are no restrictions in regards to verifying whether their lengths are equal, as is to be expected.

Due to the nature of how the protocol was built, we recommend checking that their lengths are equal to one another as well as checking that the length of one of them is equal to 2.

```
function _swap(
    address custodian,
    address[] calldata tokens,
    uint256[] calldata amounts,
    uint64 deadline,
    bytes calldata signature
) internal onlyWhitelisted(custodian) returns (bool) {
    require(tokens.length == amounts.length, "Array size mismatch");
    require(tokens.length == 2, "Array with inadequate size");
    // ...
```

This issue has been properly resolved in commit 3be1183.

4.2 Gas optimization in reused operation

In `swapTokenToNative()` the `amounts[1].sub(feeAmount)` call is used three times, performing unnecessary function calls and subsequent operations. In such cases, we recommend caching the operation that is to be reused and using the cached value instead.

```
// ...
    uint256 amount_after_fee = amounts[1].sub(feeAmount);
```

```
    IERC20(tokens[1]).safeTransferFrom(custodian, address(this),
    amount_after_fee);
    IWETH(WETH).withdraw(amount_after_fee);
    payable(msg.sender).transfer(amount_after_fee);
    // ...
```

This issue has been addressed in commit 3be1183.

4.3 Unnecessary `safeMath` functions call

The `safeMath` library starting from the 0.8.0 version of solidity does not implement additional checks for mathematical operations.

```
function add(uint256 a, uint256 b) internal pure returns (uint256) {
        return a + b;
    }
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return a - b;
    }
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        return a * b;
    }
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return a / b;
    }
```

You can avoid importing unnecessary library code as well as unnecessary calls to these functions to optimize the amount of gas used.

5 Audit Results

At the time of our audit, the code was not deployed to mainnet.

During our audit, we discovered four findings. Of these, 1 was of low risk and 2 were suggestions (Informational). Router Protocol acknowledged all findings and implemented fixes.

5.1 Disclaimers

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 additional code added to the assessed project after the audit version of 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 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.

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.