Zellic

Prepared for
Alex Qin
WOOFI

Prepared by
Kuilin Li
Vlad Toie
Zellic

April 16, 2024

WooPool

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 WOOFI from April 8th to April 15th, 2024. During this engagement, Zellic reviewed WooPool'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:

- How does WOOFI calculate the pricing for the tokens that are being swapped?
- What are the assumptions that the WooCrossChainRouterV4 makes about the StarGate interaction?
- Are there any potential griefing vectors in any of the swapping functionalities?
- How are the ERC-20 and native tokens handled within the routers? Are there any potential errors with the accounting of these tokens?

1.3. Non-goals and Limitations

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

- Front-end components
- Infrastructure relating to the project
- Key custody
- Admin management of the functions that set prices and other essential parameters

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 WooPool contracts, we discovered nine findings. No critical issues were found. Two findings were of high impact, three were of medium impact, two were of low impact, and the remaining findings were informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for WOOFI's benefit in the Discussion section (4. ↗).

Breakdown of Finding Impacts

Impact Level	Count
🟥 Critical	0
🟧 High	2
🟨 Medium	3
🟩 Low	2
◼ Informational	2

2. Introduction

2.1. About WooPool

WOOFI contributed the following description of WooPool:

> WOOFi is a decentralized exchange that bridges the deep liquidity of centralized exchanges on chain. This enables DeFi traders to swap with size and maximize their profits through the lowest swap fee and minimal slippage.

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:

WooPool Contracts

Repository	https://github.com/woonetwork/WooPoolV2 ↗
Version	WooPoolV2: `ef11dffe97bfe39fd1322756e59cd908ee15aea3`
Programs	• WooPPV2 • WooRouterV2 • WooCrossChainRouterV4 • WoooracleV2_2
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-weeks. The assessment was conducted over the course of one calendar week.

Contact Information

The following project manager was associated with the engagement:

Chad McDonald
Engagement Manager
chad@zellic.io ↗

The following consultants were engaged to conduct the assessment:

Kuilin Li
Engineer
kuilin@zellic.io ↗

Vlad Toie
Engineer
vlad@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

April 8, 2024	Kick-off call
April 8, 2024	Start of primary review period
April 15, 2024	End of primary review period

3. Detailed Findings

3.1. Successful swaps do not update spread

Target	WooPPV2		
Category	Business Logic	**Severity**	High
Likelihood	High	**Impact**	High

Description

When a swap occurs, the pool calls into the oracle to post a new price for the affected pairs:

```
// function _sellBase( ... ) {
IWooracleV2_2(wooracle).postPrice(baseToken, uint128(newPrice));

// function _sellQuote( ... ) {
IWooracleV2_2(wooracle).postPrice(baseToken, uint128(newPrice));

// function _swapBaseToBase( ... ) {
IWooracleV2_2(wooracle).postPrice(baseToken1, uint128(newBase1Price));
IWooracleV2_2(wooracle).postPrice(baseToken2, uint128(newBase2Price));
```

However, the spread is not updated. This is economically unsafe, since the spread after a trade with a large price impact is assumed to be the same as the spread before the large trade.

Impact

Users who wish to do large swaps may see better execution if they break up their swap into smaller swaps. Also, large swaps can cause the pool to lose an undue amount of value to arbitrage due to an overestimation of how accurate the current recorded price is.

Recommendations

Update the spread in addition to the price when a trade succeeds. Ensure that the new spread for the pair accurately reflects the uncertainty in price that would occur on the centralized end if such a swap was executed there.

Also, consider that a large swap does not just affect the price and spread of the pairs it touches, because of the hub-and-spoke model of the pool. For instance, if a user is trying to buy a large amount of the quote token at once, the best way to execute that would be to externally swap assets until they hold a basket of base assets similar in proportion to the reserves multiplied by each base asset's coefficient and then provide those base assets to the pool. Providing any proportion of base assets other than the most optimal one leads to poorer execution and the creation of value that arbitrageurs can then claim.

Remediation

This issue has been acknowledged by WOOFI, and a fix was implemented in commit 8b086a35 ↗.

3.2. Griefing potential over leftover tokens

Target	WooCrossChainRouterV4		
Category	Coding Mistakes	**Severity**	High
Likelihood	Low	**Impact**	High

Description

The WooCrossChainRouterV4 contract handles cross-chain swaps over StarGate. For most of its actions, it charges a fee in the form of the token that has been offered for the cross-chain operation. To retrieve these fee tokens, someone has to call the `claimFee` function, which rudimentarily transfers the entire balance of the specified `token` to the `feeAddr`, a state variable that can be set by the contract owner.

```
function claimFee(address token) external nonReentrant {
    require(feeAddr != address(0), "WooCrossChainRouterV4: !feeAddr");
    uint256 amount = _generalBalanceOf(token, address(this));
    if (amount > 0) {
        if (token == ETH_PLACEHOLDER_ADDR) {
            TransferHelper.safeTransferETH(feeAddr, amount);
        } else {
            TransferHelper.safeTransfer(token, feeAddr, amount);
        }
    }
}
```

Under normal circumstances, this function is completely safe, as the `feeAddr` is a trusted address, and the contract is not expected to hold any tokens, as most of its operations are to be performed atomically. However, if the contract is left with some tokens, as it is the case when a cross-chain operation fails (i.e due to StarGate failure), the `claimFee` function can be called by anyone to retrieve these tokens.

Impact

This can be considered a griefing vector, as the contract owner might not be aware of the leftover tokens and might not be able to retrieve them in a timely manner, leading to a potential loss of funds for the user that initiated the cross-chain operation.

Note that the cross-chain router ends up holding assets if `sgReceive` reverts, which can happen due to a gas attack if any of the tokens supported by the swap router are compromised. In the StarGate contract, if `sgReceive` reverts, the call can be retried by anyone, and before the call is retried, the cross-chain router holds the funds.

Recommendations

We recommend opting for either of the two approaches to mitigate this issue:

1. Only allow the `claimFee` function to be called by the contract owner or a trusted party. Even though this does not completely solve the issue, it reduces the potential attack surface.
2. Implement a cumulative fee mechanism, where the fee is stored in a separate contract variable, such as a mapping, upon each cumulative operation. This way, even though the fee can still be claimed by anyone, any leftover tokens would not be accounted for in the variable, so they would not be incorrectly sent out. If tokens truly get stuck, the contract owner can still retrieve them at any time via the `inCaseTokenGotStuck` function.

Remediation

This issue has been acknowledged by WOOFI.

Additionally, the WOOFI team has stated that:

> Not need to fix. The feeAddr can only be set by contract owner, which is a multi-sig safe. Even if some leftover tokens are there, the typical rescue method is also we claim the leftover tokens and send back to our user; which works well no matter whether attackers call claimFee or not.

3.3. The fallback function can collide with selectors

Target	WooracleV2_2		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	Low	**Impact**	Medium

Description

The WooracleV2_2 contract contains a fallback `onlyAdmin` function that takes in compressed data, in order for the admin to be able to update prices without paying as much gas:

```
fallback (bytes calldata _input) external onlyAdmin
    returns (bytes memory _output) {
  /*
      2 bit: 0: post prices,
             1: post states,
             2: post prices with local timestamp
             3: post states with local timestamp
      6 bits: length

      post prices:
         [price] -->
            base token: 8 bites (1 byte)
            price data: 32 bits = (27, 5)
  [...]
```

However, it is possible that the automatically generated calldata to execute this fallback function happens to match the selector of an existing function. Since the caller is an admin, this can cause undesirable effects depending on the function that is inadvertently selected.

Impact

A price update that is intended to invoke the fallback function can instead accidentally call any other function, passing in garbage for calldata.

The probability of this happening is very low, but price updates are not random, and an attacker may be able to manipulate prices and volume on the open market to increase the chance of this happening and try to control the input calldata to the resulting call.

For example, if any incorrect price is posted to a real pair due to the call being interpreted as `postPrice`, this creates an arbitrage opportunity that is likely very large.

Recommendations

We recommend adding a new role for the bot calling the fallback function, instead of making it an admin, and then changing the fallback modifier to check for that role instead of `onlyAdmin`.

This would solve this issue because even if the call collides with any function selectors, since the bot is not an admin, that mistaken call will not be made with any higher privileges than a random EOA, so it will either do something the unprivileged attacker can do anyways more directly, or it will revert.

Remediation

This issue has been acknowledged by WOOFI.

Additionally, the WOOFI team has stated that:

> It's intended to save the gas. Better not to add any more admin check or function selector. Also this contract will be called in high frequency. The probability of method conflicting is so low, and could be negligible. We have less than 30 functions in Wooracle contract, so collision probability is 30/(2^32) = 0.000000006984919; We typically update our Wooracle in 5 seconds, so a collision only happen once every 1000,000,000 seconds , that is 31 years: https://calculat.io/en/date/seconds/1000000000. From an engineering perspective: we utilize this zip fallback function to save calldata's gas consumption, so it's impossible to add another plain 4 bytes to only avoid collision. Even with collusion, our offline script can catch the tx failure and resend it again, it won't cause any disaster.

3.4. Halving spread in base-to-base may be unsafe

Target	WooPPV2		
Category	Business Logic	**Severity**	Medium
Likelihood	Medium	**Impact**	Medium

Description

In `_swapBaseToBase`, a base-to-base swap is executed by dividing the max spread of the two pairs, base-to-quote and quote-to-second-base, by two:

```
function _swapBaseToBase(
    address baseToken1,
    address baseToken2,
    uint256 base1Amount,
    uint256 minBase2Amount,
    address to,
    address rebateTo
) private nonReentrant whenNotPaused returns (uint256 base2Amount) {
    // [...]

    uint64 spread = _maxUInt64(state1.spread, state2.spread) / 2;
    uint16 feeRate = _maxUInt16(tokenInfos[baseToken1].feeRate,
    tokenInfos[baseToken2].feeRate);

    state1.spread = spread;
    state2.spread = spread;

    uint256 newBase1Price;
    (quoteAmount, newBase1Price) = _calcQuoteAmountSellBase(baseToken1,
    base1Amount, state1);

    // [...]

    uint256 newBase2Price;
    (base2Amount, newBase2Price) = _calcBaseAmountSellQuote(baseToken2,
    quoteAmount, state2);

    // [...]
}
```

Note that the swap-fee logic has been omitted from the above snippet to simplify the math below. Also, note that `_sellBase` and `_sellQuote` directly pass the `state` into the `_calc*` functions.

Assume that a user is swapping token A for token Q, which is the quote token, and then subsequently token Q for token B. Leaving out constant factors like decimals, let n_A be the notional amount of A, defined as $\Delta A * p_A$ where ΔA is the amount of token A swapped and p_A is the price.

With this, according to `_calcQuoteAmountSellBase`, for the sell base we have $\gamma = \Delta A * p_A * k_A = n_A * k_A$ and then $\Delta Q = \Delta A * p_A * (1 - \gamma - s_A) = n_A(1 - n_A k_A - s_A)$. Note that this is a quadratic polynomial in n_A where the only occurrence of the spread is $-s_A$ as a constant linear term in the coefficient for the first degree n_A.

Similarly, according to `_calcBaseAmountSellQuote`, we have (note, this is a different γ) $\gamma = \Delta Q * k_B$ and then $\Delta B = \Delta Q * (1 - \gamma - s_B)/p_B$. So, again substituting, we have $n_B = \Delta Q * (1 - \Delta Q * k_B - s_B)$.

Now, if we substitute in ΔQ, the result becomes a fourth-degree polynomial in n_A.

Now on the other hand, if they executed `_swapBaseToBase` instead of base-to-quote and then quote-to-base, then according to the above code snippet, the calculation will be the same except $s'_A = s'_B = \texttt{max}(s_A, s_B)/2$ — let this just be s'. This means that the same expansion will follow in the calculation of n'_B in terms of n_A, and so the difference $n'_B - n_B$ will cancel out all the terms that do not depend on any s.

So, with that simplification, the terms of n_B that include some s will sum to $\Delta Q * (1 - \Delta Q * k_B - s_B) = (1 - s_B)\Delta Q - k_B(\Delta Q)^2 = (1 - s_B)(1 - s_A)n_A - (1 - s_B)k_A n_A - k_B(1 - s_A)^2 n_A^2$.

So, whether the base-to-base return is better than the base-to-quote-to-base case clearly depends on the exact quantities involved, and there are examples for both. As a quick example, if we assume the spread for token B were zero, then

- The first term is always higher, because $(1 - s_A/2)^2 > (1 - s_A)$.
- The second term cancels out because there is no dependence on s_A.
- The third term is always lower, because $-k_B(1 - s_A/2)^2 < -k_B(1 - s_A)^2$ (note that the domain of s_A is from zero to one, and the breakeven points are the roots at zero and 1.33).

This means that which is bigger depends entirely on whether k_B is larger or smaller, because it will control whether the first or third term dominates and the liquidity coefficient is a completely free parameter.

Impact

Due to nonlinearity, the approximation `spread = _maxUInt64(state1.spread, state2.spread) / 2` may produce a large error if there is a large difference in oracle parameters, especially the liquidity coefficient k, such that users may get better execution doing a two-stage swap over a direct swap — whether that be base-to-base-to-quote instead of base-to-quote, or a base-to-quote-to-base instead of a base-to-base depending on the direction of the discrepancy.

Recommendations

We recommend not improving the spread in the base-to-base case and using the original spreads such that the only savings of base-to-base compared to base-to-quote-to-base is that the swap fee is not charged twice.

Remediation

This issue has been acknowledged by WOOFI, and a fix was implemented in commit `879afe26` ↗.

3.5. Chainlink data staleness

Target	WooracleV2_2		
Category	Business Logic	**Severity**	Medium
Likelihood	Low	**Impact**	Medium

Description

The Wooracle contract relies on Chainlink as one of its third-party pricing sources. The `latestRoundData` method is used to retrieve data, including token prices. However, the contract does not sufficiently check for stale data, which can result in inaccurate pricing information.

```
function _cloPriceInQuote(address _fromToken, address _toToken)
    internal
    view
    returns (uint256 refPrice, uint256 refTimestamp)
{
    address baseOracle = clOracles[_fromToken].oracle;

    // NOTE: Only for chains where chainlink oracle is unavailable
    // if (baseOracle == address(0)) {
    // return (0, 0);
    // }
    require(baseOracle != address(0), "WooracleV2_2: !oracle");

    address quoteOracle = clOracles[_toToken].oracle;
    uint8 quoteDecimal = clOracles[_toToken].decimal;


    (, int256 rawBaseRefPrice, , uint256 baseUpdatedAt, )
    = AggregatorV3Interface(baseOracle).latestRoundData();
    (, int256 rawQuoteRefPrice, , uint256 quoteUpdatedAt, )
    = AggregatorV3Interface(quoteOracle).latestRoundData();
    // ...
}
```

Impact

Should the data returned by Chainlink be stale, the health-factor calculation would be incorrect, which could in turn affect the entire protocol.

Recommendations

We recommend adding checks for each of the returned values to ensure that the data is not stale.

```
function _cloPriceInQuote(address _fromToken, address _toToken)
    internal
    view
    returns (uint256 refPrice, uint256 refTimestamp)
{
    address baseOracle = clOracles[_fromToken].oracle;

    // NOTE: Only for chains where chainlink oracle is unavailable
    // if (baseOracle == address(0)) {
    // return (0, 0);
    // }
    require(baseOracle != address(0), "WooracleV2_2: !oracle");

    address quoteOracle = clOracles[_toToken].oracle;
    uint8 quoteDecimal = clOracles[_toToken].decimal;


    (
       uint80 baseRoundID,
        int256 rawBaseRefPrice,
       uint baseStartedAt,
        uint256 baseUpdatedAt,
       uint80 baseAnsweredInRound
    ) = AggregatorV3Interface(baseOracle).latestRoundData();
    (
       uint80 quoteRoundID,
        int256 rawQuoteRefPrice,
       uint quoteStartedAt,
        uint256 quoteUpdatedAt
       uint80 quoteAnsweredInRound
    ) = AggregatorV3Interface(quoteOracle).latestRoundData();

    require(
       baseUpdatedAt != 0,
       "base round is not complete"
    );

    require(
        baseAnsweredInRound >= baseRoundID,
        "base stale data"
    );

    require(
       quoteUpdatedAt != 0,
       "quote round is not complete"
    );

    require(
        quoteAnsweredInRound >= quoteRoundID,
        "quote stale data"
    );
    // ...
}
```

Remediation

This issue has been acknowledged by WOOFI.

Additionally, the WOOFI team has stated that:

> Checking the staleness onchain costs more gas than necessary (regarding the TTL is typically 24 hours). However, we add an offchain script monitor for chainlink staleness (as well as L2 sequencer status), if any oracle got expired, we pause the WooPP immediately. So essentially, we're having the staleness monitor now, but in offchain.

3.6. Sandwich attack can affect base-to-base swap fee

Target	WooPPV2		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

Normally, sandwich attacks on swaps through the WooPPV2 pool are mitigated by the minimum output amount checks, such that even if a sandwich attack occurs, the user is still left with an amount of tokens that they are happy with.

This means that, assuming normal operation, for every swap, the fee charged (which is always denominated in the quote token) should always be a reasonable amount of real value, since if it was not, the user would be left with an amount of output that they would be unhappy with, which would have violated the minimum output requirement.

However, this assumption fails for `_swapBaseToBase`. When a base-to-base swap occurs, the first base is first swapped for the quote token, then the fee is charged as a percentage of the intermediary quote amount, and finally the quote amount is swapped for the second base. The amount of quote tokens received in the middle is not a quantity that the caller cares about, and it is not checked — only the output amount of the base 2 token is checked.

Impact

If a large base-to-base swap exists, an attacker or the caller can sandwich this swap to artificially increase or decrease the amount of quote tokens paid as fees from the reserve to the fee recipient.

For example, let us say A and B are base tokens and the quote token is Q. On the mempool, there is a large swap from A to B. If the swap occurred normally, 10 Q tokens will be charged as swap fees, corresponding to a fixed percentage `feeRate` of the value.

However, if an attacker sandwiches this swap in order to artificially temporarily increase the prices of both A and B by the same factor during the transaction, then the `quoteAmount` in the middle of the swap will be increased, so more than 10 Q tokens will be paid to the fee recipient. Since the prices are increased by the same factor, the swapper still gets the same amount of tokens back (the same value minus the fixed percentage), but now more quote tokens have been transferred from the reserve to the fee recipient.

Recommendations

There is not really a way to remediate this issue, because the price of the quote token should fluctuate according to intra-block transactions, and as long as the fee is denominated in the quote token, there is no way to independently calculate the correct amount of fee to charge for a base-to-base swap if prices are not trusted.

However, note that mitigating Finding 3.1. ↗ also mitigates this issue, because if large transactions always increase the spread, it becomes prohibitively expensive to temporarily change any prices significantly. If a sandwich attack waits for a price update before finishing, so that the spread goes back down to normal, then that adds in the risk that other arbitrageurs step in and take the capital used in the attack.

Remediation

This issue has been acknowledged by WOOFI, and a fix was implemented in commit 8b086a35 ↗.

3.7. Arbitrary calldata in `externalSwap` may be unsafe

Target	WooRouterV2		
Category	Protocol Risks	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

When a user performs an external swap, either normally or through the cross-chain router, they can arbitrarily specify the calldata sent to the external swap service:

```
function externalSwap(
    address approveTarget,
    address swapTarget,
    address fromToken,
    address toToken,
    uint256 fromAmount,
    uint256 minToAmount,
    address payable to,
    bytes calldata data
) external payable override nonReentrant returns (uint256 realToAmount) {
    // [...]
    require(isWhitelisted[swapTarget], "WooRouter: SWAP_TARGET_NOT_ALLOWED");

    // [...]

    _internalFallbackSwap(approveTarget, swapTarget, fromToken, fromAmount,
    data);

    // [...]
}

function _internalFallbackSwap(
    address approveTarget,
    address swapTarget,
    address fromToken,
    uint256 fromAmount,
    bytes calldata data
) private {
    require(isWhitelisted[approveTarget], "WooRouter:
    APPROVE_TARGET_NOT_ALLOWED");
    require(isWhitelisted[swapTarget], "WooRouter: SWAP_TARGET_NOT_ALLOWED");
    if (fromToken != ETH_PLACEHOLDER_ADDR) {
        TransferHelper.safeTransferFrom(fromToken, msg.sender, address(this),
    fromAmount);
        TransferHelper.safeApprove(fromToken, approveTarget, fromAmount);
        (bool success, ) = swapTarget.call{value: 0}(data);
        TransferHelper.safeApprove(fromToken, approveTarget, 0);
        require(success, "WooRouter: FALLBACK_SWAP_FAILED");
    } else {
        require(fromAmount <= msg.value, "WooRouter: fromAmount_INVALID");
        (bool success, ) = swapTarget.call{value: fromAmount}(data);
        require(success, "WooRouter: FALLBACK_SWAP_FAILED");
    }
}
```

Even though the swap target and the approve target are whitelisted, the calldata is not parsed to ensure that the user is, in fact, calling a swap function.

Impact

If a whitelisted external-swap target contract has any functions that cause unrelated side effects, those side effects can be manipulated in arbitrary ways by users executing false swaps.

For example, if a swap target implements a `claimReward` function that claims rewards based on a caller's swap volume or a `setDelegate` function to allow the caller to grant permissions to another address to take some action on behalf of the caller, or if the swap target is also an ERC-20 token and has a `transfer` function — in all of these cases, a user can profit off of a fake swap.

Even if a swap target does not currently implement such a function, if it is upgradable, a future iteration of it may implement a risky function. For example, consider the case where a swap target is exploited, but the exploit is fixed and funding is secured to reimburse victims, and each affected user can call `claimShare` on the contract to claim what they lost. If this happens, WOOFI would likely wish to manually call this on behalf of the pool and further divide the share among its users. However, because of this arbitrary calldata issue, they will likely be beaten by someone claiming all the tokens for themselves using a false external swap.

Recommendations

Since the swap target whitelist is already a manually managed whitelist, we recommend including the four-byte signature in the key for this whitelist so that users can only call a permitted `(swapTarget, function)` pair. This greatly reduces the chance that the whitelist allows for an unexpected call, even if the swap target was upgraded to add additional functionality.

Remediation

This issue has been acknowledged by WOOFI.

Additionally, the WOOFI team has stated that:

> Will consider adding a function selector check in later version. But still, the official doc of 1inch integration is not having this check, and we're concerned that it may not have a fixed function to enforce.

3.8. Bridge fee only charged on convenience swap

Target	WooCrossChainRouterV4		
Category	Business Logic	**Severity**	Informational
Likelihood	High	**Impact**	Informational

Description

The function `crossSwap` is called on WooCrossChainRouterV4 to initiate a cross-chain swap:

```
function crossSwap(
    uint256 refId,
    address payable to,
    SrcInfos memory srcInfos,
    DstInfos calldata dstInfos,
    Src1inch calldata src1inch,
    Dst1inch calldata dst1inch
) external payable whenNotPaused nonReentrant {
    // [...]

    uint256 fee = 0;

    // Step 1:  transfer

    // [...]

    // Step 2:  local swap by 1inch router
    if (srcInfos.fromToken != srcInfos.bridgeToken) {
        TransferHelper.safeApprove(srcInfos.fromToken, address(wooRouter),
    srcInfos.fromAmount);
        if (src1inch.swapRouter != address(0)) {
            // external swap via 1inch
            bridgeAmount = wooRouter.externalSwap(
                src1inch.swapRouter,
                src1inch.swapRouter,
                srcInfos.fromToken,
                srcInfos.bridgeToken,
                srcInfos.fromAmount,
                srcInfos.minBridgeAmount,
                payable(address(this)),
                src1inch.data
            );
            fee = (bridgeAmount * srcExternalFeeRate) / FEE_BASE;
        } else {
            // swap via WOOFi
            bridgeAmount = wooRouter.swap(
                srcInfos.fromToken,
                srcInfos.bridgeToken,
                srcInfos.fromAmount,
                srcInfos.minBridgeAmount,
                payable(address(this)),
                to
            );
        }
    } else {
        require(
            srcInfos.fromAmount == srcInfos.minBridgeAmount,
            "WooCrossChainRouterV4: !srcInfos.minBridgeAmount"
        );
        bridgeAmount = srcInfos.fromAmount;
    }

    // Step 3: deduct the swap fee
    bridgeAmount -= fee;
    require(bridgeAmount >= srcInfos.minBridgeAmount, "WooCrossChainRouterV4:
    !srcInfos.minBridgeAmount");

    // Step 4: cross chain swap by StarGateRouter
    _bridgeByStargate(refId, to, msgValue, bridgeAmount, srcInfos, dstInfos,
    dst1inch);

    // [...]
}
```

Note that the bridge charges an additional `fee` of the `srcExternalFeeRate` only when an external swap is done on the source chain before bridging.

However, a user can just do the same swap externally before calling the bridge and avoid this fee. Note that the external swap itself could charge fees, and if it does, those fees would be on top of the `fee` charged here.

Impact

This fee can be avoided by users externally doing the swap themselves.

Recommendations

We recommend charging a fee to use the bridge, instead of charging a fee to transactionally externally swap before using the bridge and allowing users who notice this discrepancy to essentially use the bridge for free. This fee model better aligns the charged fee with what the users are deriving value from.

Remediation

This issue has been acknowledged by WOOFI.

Additionally, the WOOFI team has stated that:

> It is by design. And we only charge the fee for external swap in destination chain, and decided in the meeting to not charge extra bridge fee.

3.9. Extra WETH change claimable by next caller

Target	WooRouterV2		
Category	Business Logic	**Severity**	Informational
Likelihood	High	**Impact**	Informational

Description

When the WooRouterV2 is used to perform a swap, the user may pay in native ETH:

```
function swap(
    address fromToken,
    address toToken,
    uint256 fromAmount,
    uint256 minToAmount,
    address payable to,
    address rebateTo
) external payable override nonReentrant returns (uint256 realToAmount) {
    require(fromToken != address(0), "WooRouter: !fromToken");
    require(toToken != address(0), "WooRouter: !toToken");
    require(to != address(0), "WooRouter: !to");

    bool isFromETH = fromToken == ETH_PLACEHOLDER_ADDR;
    bool isToETH = toToken == ETH_PLACEHOLDER_ADDR;
    fromToken = isFromETH ? WETH : fromToken;
    toToken = isToETH ? WETH : toToken;

    // Step 1: transfer the source tokens to WooRouter
    if (isFromETH) {
        require(fromAmount <= msg.value, "WooRouter: fromAmount_INVALID");
        IWETH(WETH).deposit{value: msg.value}();
        TransferHelper.safeTransfer(WETH, address(wooPool), fromAmount);
    } else {
        TransferHelper.safeTransferFrom(fromToken, msg.sender,
    address(wooPool), fromAmount);
    }

    // [...]
```

However, if the caller accidentally specifies a `fromAmount` that is less than, instead of being equal to, the `msg.value` sent with the call, that resulting WETH stays in the contract.

Impact

Any extra WETH becomes stuck in the contract. Similarly, if value is accidentally sent but `fromToken` is not set to `ETH_PLACEHOLDER_ADDR`, then the native ETH becomes stuck in the contract. In these cases, the owner needs to manually call `inCaseTokenGotStuck` to retrieve the value.

Recommendations

If it is intended that any extra ETH, wrapped or not, that is mistakenly left in the contract should be treated as a donation to the protocol, then it should explicitly be transferred to the pool reserves or the fee recipient. On the other hand, if the funds will be returned to the user, it would save effort and increase trust to just require that `fromAmount == msg.value` in the `isFromEth` case, and `msg.value == 0` in the other branch, so that the transaction reverts if this mistake is made.

Remediation

This issue has been acknowledged by WOOFI, and a fix was implemented in commit 11183887 ↗.

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. Function `skimMulTokens` does not work

In WooPPV2, the `onlyAdmin` function `skim` transfers to the owner any balance above the reserve of the token, which includes fees and tokens erroneously sent to the contract:

```
function skim(address token) public nonReentrant onlyAdmin {
    TransferHelper.safeTransfer(token, owner(), balance(token)
    - tokenInfos[token].reserve);
}
```

It also has a function `skimMulTokens` that calls this for multiple tokens:

```
function skimMulTokens(address[] memory tokens)
    external nonReentrant onlyAdmin {
    unchecked {
        uint256 len = tokens.length;
        for (uint256 i = 0; i < len; i++) {
            skim(tokens[i]);
        }
    }
}
```

However, the `nonReentrant` modifier is on both of these functions, which means that `skimMulTokens` will always revert due to the reentrancy guard.

This issue has been acknowledged by WOOFI, and a fix was implemented in commit `c19894f9` ↗.

4.2. Throwing Panic is not recommended

The `receive` hook for WooPPV2 uses `assert` to ensure that it does not accept native ETH from an unknown source:

```
receive() external payable {
    // only accept ETH from WETH or whitelisted external swaps.
    assert(msg.sender == WETH || isWhitelisted[msg.sender]);
}
```

However, in Solidity, a failed `assert` will throw a Panic exception. According to the Solidity docs,

> Assert should only be used to test for internal errors, and to check invariants. Properly functioning code should never create a Panic, not even on invalid external input. If this happens, then there is a bug in your contract which you should fix. Language analysis tools can evaluate your contract to identify the conditions and function calls which will cause a Panic.

So, to ensure compatibility with automated language-analysis tools and to better align with the standard for error types, this check should be replaced with a `require`, with or without a specific error message.

This issue has been acknowledged by WOOFI, and a fix was implemented in commit `39deb1cd` ↗.

4.3. The WooPPV2 contract doesn't need to be an admin

When the WooPPV2 contract makes a swap, it calls `postPrice` on the WooracleV2_2 to post the new price for future swaps:

```
IWooracleV2_2(wooracle).postPrice(baseToken, uint128(newPrice));
```

Here is the implementation of `postPrice`:

```
function postPrice(address _base, uint128 _price) external onlyAdmin {
    infos[_base].price = _price;
    if (msg.sender != wooPP) {
        timestamp = block.timestamp;
    }
}
```

Since `postPrice` is `onlyAdmin`, this means the `wooPP` address must be set as an admin on the oracle contract. However, this means that the `wooPP` can call any other admin function, including the fallback function.

There is currently no way for an unprivileged caller to cause WooPPV2 to call the oracle in an unexpected manner without it also reverting. But, since the fallback function accepts many function selectors, a future version of either contract could introduce a potential negative interaction where the pool calls an external contract using a user-controlled address, for instance to transfer an unknown ERC20 token or to make a callback call, and an attacker unexpectedly passes in the address of the oracle instead of the token or receiver address.

Since `postPrice` already checks if the caller is `wooPP`, it wouldn't cost any more gas to repeat the body of the `onlyAdmin` check in the function with an extra case for `wooPP`. Alternatively, another function can be added that only allows `wooPP` to call it, that posts the price without resetting the timestamp. Either of these two strategies would make it so that `wooPP` cannot call any of the other `onlyAdmin` functions.

This issue has been acknowledged by WOOFI, and the fix has been implemented in commit ea79a0d ↗

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: WooCrossChainRouterV4.sol

Function: `claimFee(address token)`

Allows claiming the cumulative fees of the token.

Inputs

- `token`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Assumes there is a token to claim.
  - **Impact**: The token to claim.

Branches and code coverage

Intended branches

- Assume the balance corresponds to the cumulative sum of the fees. This is not always the case, as in the case of bridging transactions that are not successful. These tokens can be griefed to the `feeAddr` address.
  - ☐ Test coverage
- Transfer the balance of the token to the fee address.
  - ☑ Test coverage

Negative behavior

- Caller is a service admin.
  - ☑ Negative test
- Negative behavior should be what the function requires.
  - ☐ Negative test

Function: `crossSwap(uint256 refId, address payable to, SrcInfos srcInfos, DstInfos dstInfos, Src1inch src1inch, Dst1inch dst1inch)`

Handles the cross-chain swap via StarGate.

Inputs

- `refId`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None — assumed to be unique but not enforced.
  - **Impact**: The reference ID of the transaction.
- `to`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None — assumed to be a valid address.
  - **Impact**: The address to send the bridged tokens to.
- `srcInfos`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Some of its parameters are checked to be valid.
  - **Impact**: The source information.
- `dstInfos`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Some of its parameters are checked to be valid.
  - **Impact**: The destination information.
- `src1inch`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: Checked that `.swapRouter` is not `address(0)`.
  - **Impact**: The `1inch` router to use for the swap on the source chain.
- `dst1inch`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None at this level.
  - **Impact**: The `1inch` router to use for the swap on the destination chain.

Branches and code coverage

Intended branches

- Ensure that `dstInfos.chainId` is a valid chain ID. Currently not enforced.
  - □ Test coverage
- Ensure that `dstInfos.toToken != 0` and `dstInfos.toToken != sgInfo.sgETHs(dstInfos.chainId)`.
  - ☑ Test coverage
- Ensure that `srcInfos.bridgeToken` corresponds to the `sgPoolIds` of the source chain.
  - ☑ Test coverage
- Ensure that `dstInfos.bridgeToken` corresponds to the `sgPoolIds` of the destination chain.
  - ☑ Test coverage
- Ensure that `msg.value` covers for the cross-chain swap if native tokens are used (`srcInfos.fromToken == ETH_PLACEHOLDER_ADDR`).

- ☑ Test coverage
- Transfer the necessary `srcInfos.fromAmount` tokens to this contract.
  - ☑ Test coverage
- Approve the `srcInfos.fromAmount` tokens to the `wooRouter` in case they are to be swapped.
  - ☑ Test coverage
- Forward the bridge call over to the StarGate router.
  - ☑ Test coverage

Negative behavior

- Should not be callable when the contract is paused. Enforced through the `whenNotPaused` modifier.
  - ☑ Negative test
- Should not be reentrant. Enforced through the `nonReentrant` modifier.
  - ☑ Negative test

Function: `inCaseTokenGotStuck(address stuckToken)`

Function that handles transferring stuck tokens to the owner.

Inputs

- `stuckToken`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None. Checked whether it is `ETH_PLACEHOLDER_ADDR`.
  - **Impact**: The token to be transferred.

Branches and code coverage

Intended branches

- Transfer stuck tokens to the owner.
  - ☑ Test coverage
- Decrease `stuckToken` balance of the contract to zero.
  - ☑ Test coverage
- Increase `stuckToken` balance of the owner by the amount of `stuckToken` in the contract.
  - ☑ Test coverage

Negative behavior

- Should not be callable by anyone other than the owner.
  - ☑ Negative test

Function: `pause()`

Allows the owner to pause the contract.

Branches and code coverage

Intended branches

- Assumes that the contract is not paused in the first place.
  - □ Test coverage
- Pause the contract.
  - ☑ Test coverage

Negative behavior

- Should not be callable by anyone other than the owner.
  - ☑ Negative test

Function: `sgReceive(uint16, bytes, uint256, address bridgedToken, uint256 amountLD, bytes payload)`

Function that handles receiving via StarGate.

Inputs

- `srcChainId`
  - **Control**: Fully controlled by the caller. Currently not used.
  - **Constraints**: None. Should check that it is a valid chain ID for the source chain.
  - **Impact**: The source chain where the cross-chain transaction originated.
- `srcAddress`
  - **Control**: Fully controlled by the caller. Currently not used.
  - **Constraints**: None. Should check that it corresponds to the WoocrossChain-RouterV4 contract on the source chain.
  - **Impact**: The address of the originating contract on the source chain.
- `nonce`
  - **Control**: Fully controlled by the caller. Currently not used.
  - **Constraints**: None. Should be used to prevent replay attacks.
  - **Impact**: The nonce of the transaction.
- `bridgedToken`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None. Assumed to be a token that can be bridged.
  - **Impact**: The token that was bridged.
- `amountLD`

- **Control**: Fully controlled by the caller.
- **Constraints**: None. Assumed to be the amount of the bridged token. When called by StarGate, StarGate ensures that this amount the token has been transferred to the contract by StarGate as a result of the cross-chain call.
- **Impact**: The amount of the bridged token.

- `payload`
  - **Control**: Fully controlled by the caller.
  - **Constraints**: None. Assumed to be the payload of the cross-chain transaction.
  - **Impact**: The payload of the cross-chain transaction.

Branches and code coverage

Intended branches

- Ensure that `srcChainId` is a valid chain ID. Currently not enforced.
  - ☐ Test coverage
- Ensure that `srcAddress` corresponds to the WoocrossChainRouterV4 contract on the source chain. Currently not enforced.
  - ☐ Test coverage
- Ensure that `nonce` is used to prevent replay attacks. Currently not enforced.
  - ☐ Test coverage
- If `bridgedToken` is `SGETH`, handle the native received.
  - ☑ Test coverage
- If `bridgedToken` is not `SGETH`, handle the ERC-20 received.
  - ☑ Test coverage

Negative behavior

- Should not allow anyone other than the StarGate router to call this function.
  - ☑ Negative test

Function: `unpause()`

Allows the owner to unpause the contract.

Branches and code coverage

Intended branches

- Assumes that the contract is paused in the first place.
  - ☐ Test coverage
- Unpause the contract.
  - ☑ Test coverage

Negative behavior

- Should not be callable by anyone other than the owner.
  - ☑ Negative test

Function: `_bridgeByStargate(uint256 refId, address payable to, uint256 msgValue, uint256 bridgeAmount, SrcInfos srcInfos, DstInfos dstInfos, Dst1inch dst1inch)`

Allows bridging via StarGate.

Inputs

- `refId`
  - **Control**: Fully controlled by the calling function.
  - **Constraints**: None.
  - **Impact**: The reference ID of the transaction.
- `to`
  - **Control**: Fully controlled by the calling function.
  - **Constraints**: None.
  - **Impact**: The address to send the bridged tokens to.
- `msgValue`
  - **Control**: Fully controlled by the calling function.
  - **Constraints**: None. Theoretically only used for the `native` swap case.
  - **Impact**: Cumulated `msg.value` from the calling function.
- `bridgeAmount`
  - **Control**: Fully controlled by the calling function.
  - **Constraints**: None. Assumed that these tokens have been transferred atomically (i.e., same transaction) to this contract.
  - **Impact**: The amount of the bridged token.
- `srcInfos`
  - **Control**: Fully controlled by the calling function.
  - **Constraints**: Minimal. Most are performed in the calling function.
  - **Impact**: The source information.
- `dstInfos`
  - **Control**: Fully controlled by the calling function.
  - **Constraints**: Minimal. Most are performed in the calling function.
  - **Impact**: The destination information.
- `dst1inch`
  - **Control**: Fully controlled by the calling function.
  - **Constraints**: None. Assumed to be valid.
  - **Impact**: The `1inch` router to use for the swap (if any).

Branches and code coverage

Intended branches

- If `bridgeToken` is WETH, withdraw the `bridgeAmount` from the WETH contract so that it can be used as the `msg.value`.
  - ☑ Test coverage
- If `bridgeToken` is not WETH, approve the `bridgeAmount` of the `bridgeToken` to the StarGate router. Assumes these tokens have been transferred atomically (i.e., same transaction) to this contract.
  - ☑ Test coverage
- Forward the cross-chain swap to StarGate and forward the `payload`.
  - ☑ Test coverage

Negative behavior

- Caller is a service admin.
  - ☑ Negative test
- Negative behavior should be what the function requires.
  - ☐ Negative test

Function: `_handleERC20Received(uint256 refId, address to, address toToken, address bridgedToken, uint256 bridgedAmount, uint256 minToAmount, Dst1inch dst1inch)`

Handles received ERC-20 tokens via StarGate.

Inputs

- `refId`
  - **Control**: Fully controlled by calling function.
  - **Constraints**: None.
  - **Impact**: The reference ID of the transaction.
- `to`
  - **Control**: Fully controlled by calling function.
  - **Constraints**: None.
  - **Impact**: The address to send the bridged tokens to.
- `toToken`
  - **Control**: Fully controlled by calling function.
  - **Constraints**: Checked whether it is the `bridgedToken`.
  - **Impact**: The token to send to the `to` address.
- `bridgedToken`
  - **Control**: Fully controlled by calling function.
  - **Constraints**: Checked whether it is the `toToken`.

    - **Impact**: The token that was bridged.
- `bridgedAmount`
    - **Control**: Fully controlled by calling function.
    - **Constraints**: None. Assumed to be correctly forwarded by StarGate.
    - **Impact**: The amount of the bridged token.
- `minToAmount`
    - **Control**: Fully controlled by calling function.
    - **Constraints**: None. Assumed that checks are performed in the `wooRouter`.
    - **Impact**: The minimum amount of the `toToken` to receive.
- `dst1inch`
    - **Control**: Fully controlled by calling function.
    - **Constraints**: None. Checked that it is not `address(0)`.
    - **Impact**: The `1inch` router to use for the swap.

Branches and code coverage

Intended branches

- Check that `realToAmount` is greater than `minToAmount`. Currently not checked.
    - ☐ Test coverage
- If `toToken` is the same as `bridgedToken`, transfer the bridged amount to the `to` address.
    - ☑ Test coverage
- If `toToken` is not the same as `bridgedToken`, swap the bridged token to the `toToken`, specifying the `to` address as the recipient.
    - ☑ Test coverage
- If either of the swaps did not succeed, transfer the bridged amount of tokens directly to the `to` address.
    - ☑ Test coverage
- Ensure that adequate approvals are performed before calling the swap functions.
    - ☑ Test coverage
- Ensure that approvals are removed after the swap is performed.
    - ☑ Test coverage

Negative behavior

- The source address of the cross-chain transaction should be a WooCrossChainRouter on the source chain; this is currently not enforced at any level.
    - ☐ Negative test
- Caller should be the StarGate router. This is enforced at calling function level.
    - ☑ Negative test

Function: `_handleNativeReceived(uint256 refId, address to, address toToken, uint256 bridgedAmount, uint256 minToAmount, Dst1inch dst1inch)`

Handles the receipt of native via StarGate.

Inputs

- `refId`
    - **Control**: Fully controlled by calling function.
    - **Constraints**: None.
    - **Impact**: The reference ID of the transaction.
- `to`
    - **Control**: Fully controlled by calling function.
    - **Constraints**: None.
    - **Impact**: The address to send the bridged tokens to.
- `toToken`
    - **Control**: Fully controlled by calling function.
    - **Constraints**: Checked whether it is the `ETH_PLACEHOLDER_ADDR`.
    - **Impact**: The token to send to the `to` address.
- `bridgedAmount`
    - **Control**: Fully controlled by calling function.
    - **Constraints**: None. Assumed to be correctly forwarded by StarGate.
    - **Impact**: The amount of the bridged token.
- `minToAmount`
    - **Control**: Fully controlled by calling function.
    - **Constraints**: None. Assumed that checks are performed in the `wooRouter`.
    - **Impact**: The minimum amount of the `toToken` to receive.
- `dst1inch`
    - **Control**: Fully controlled by calling function.
    - **Constraints**: None. Checked that it is not `address(0)`.
    - **Impact**: The `1inch` router to use for the swap.

Branches and code coverage

Intended branches

- Check that `realToAmount` is greater than `minToAmount`. Currently not checked.
    - ☐ Test coverage
- If the `toToken` is `ETH_PLACEHOLDER_ADDR`, directly transfer the bridged amount as the native token, then return to exit early.
    - ☑ Test coverage
- If the `toToken` is not `ETH_PLACEHOLDER_ADDR`, wrap it as WETH, then swap it to the `toToken` if required.

  - ☑ Test coverage
- If the `toToken` is not `ETH_PLACEHOLDER_ADDR`, and either of the swaps did not succeed, transfer the bridged amount as WETH to the `to` address.
  - ☑ Test coverage
- Ensure that adequate approvals are performed before calling the swap functions.
  - ☑ Test coverage
- Ensure that approvals are removed after the swap is performed.
  - ☑ Test coverage

Negative behavior

- The source address of the cross-chain transaction should be a WooCrossChainRouter on the source chain; this is currently not enforced at any level.
  - □ Negative test
- Caller should be the StarGate router. This is enforced at calling function level.
  - ☑ Negative test

5.2. Module: WooPPV2.sol

Function: `claimFee()`

Allows anyone to transfer the unclaimed fees to the fee address.

Branches and code coverage

Intended branches

- Fee transfer succeeds.
  - ☑ Test coverage

Negative behavior

- Fee address must be set.
  - □ Negative test

Function: `deposit(address token, uint256 amount)`

Allows the owner to deposit tokens into the contract.

Inputs

- `token`
  - **Control**: Fully controlled.
  - **Constraints**: Assumed that the token is a valid ERC-20 token.

    - **Impact**: The token to deposit.
- `amount`
    - **Control**: Fully controlled.
    - **Constraints**: Will fail if `msg.sender` does not have enough tokens.
    - **Impact**: The amount of tokens to deposit.

Branches and code coverage

Intended branches

- Increase the `balance` of the token for the contract.
    ☑ Test coverage
- Decrease the `balance` of the token for the caller.
    ☑ Test coverage
- Adapt the `reserve` of the token by the amount deposited.
    ☑ Test coverage

Negative behavior

- Should not be callable by anyone other than the `admin`.
    ☑ Negative test

Function: `inCaseTokenGotStuck(address stuckToken)`

Allows withdrawing any token from the contract, on the assumption that the token is stuck.

Inputs

- `stuckToken`
    - **Control**: Full control.
    - **Constraints**: None.
    - **Impact**: The token will be withdrawn from the contract.

Branches and code coverage

Intended branches

- Transfers the entire balance of a specific token to the `msg.sender` (i.e., the owner).
    ☑ Test coverage

Negative behavior

- Should not be callable by anyone other than the `owner`.
    ☑ Negative test

- Should not be called with malicious intent.
  - ☑ Negative test

Function: `migrateToNewPool(address token, address newPool)`

Allows the owner to migrate the pool to a new pool contract.

Inputs

- `token`
  - **Control**: Full control.
  - **Constraints**: Must be a valid token address.
  - **Impact**: The token will be migrated to the new pool.
- `newPool`
  - **Control**: Full control.
  - **Constraints**: Must be a valid pool address.
  - **Impact**: The pool will be migrated to the new pool.

Branches and code coverage

Intended branches

- Assumes that the all the tokens have been withdrawn from the previous pool.
  - ☐ Test coverage
- Assumes the other pool has the corresponding `tokenInfos` of this pool. Otherwise, all the reserves calculations will be wrong.
  - ☐ Test coverage
- Should make this contract not usable after migration. Currently not implemented.
  - ☐ Test coverage

Negative behavior

- Should not be callable by anyone other than the owner.
  - ☑ Negative test

Function: `_sellBase(address baseToken, uint256 baseAmount, uint256 minQuoteAmount, address to, address rebateTo)`

Allows selling base token for quote token.

Inputs

- `baseToken`
    - **Control**: Controlled by calling function.
    - **Constraints**: Checked to be different than zero.
    - **Impact**: The base token to swap from.
- `baseAmount`
    - **Control**: Controlled by calling function.
    - **Constraints**: Checked to be above the amount of `baseToken` that has to be swapped.
    - **Impact**: The amount of `baseToken` to swap.
- `minQuoteAmount`
    - **Control**: Controlled by calling function.
    - **Constraints**: Checked to be above the amount of `quoteToken` that has to be received.
    - **Impact**: The minimum expected amount of resulting quote tokens.
- `to`
    - **Control**: Controlled by calling function.
    - **Constraints**: Checked to be different than zero.
    - **Impact**: The address to send the swapped `baseToken2` to.
- `rebateTo`
    - **Control**: Controlled by calling function.
    - **Constraints**: None.
    - **Impact**: The address to send the rebate to.

Branches and code coverage

Intended branches

- Ensure that the spread changes after the swap. Currently not enforced.
    - ☐ Test coverage
- Ensure that `baseToken` is not zero or the `quoteToken`.
    - ☑ Test coverage
- Ensure that `to` is not zero.
    - ☑ Test coverage
- Ensure that the current balance minus the reserve is above `quoteAmount` (i.e., that tokens have been transferred beforehand).
    - ☑ Test coverage
- Should post the new price of `baseToken` after the swap on the `wooOracle`.
    - ☑ Test coverage
- Should calculate the swap fee and account for it in the `unclaimedFee`.
    - ☑ Test coverage
- Update the reserves of the quote and base tokens.

- ☑ Test coverage
- Transfer the resulting `quoteAmount` worth of `quoteToken` to the `to` address.
  - ☑ Test coverage

Negative behavior

- Should not be callable when the contract is paused.
  - ☑ Negative test
- Should not reenter.
  - ☑ Negative test
- Should not allow swapping between the same tokens.
  - ☑ Negative test
- Should not allow performing the swap if no tokens have been transferred beforehand. Ensured in the `require` on balance check.
  - ☑ Negative test
- Should not allow a swap if `minBaseAmount` is not met.
  - ☑ Negative test

Function: `_sellQuote(address baseToken, uint256 quoteAmount, uint256 minBaseAmount, address to, address rebateTo)`

Allows swapping from `quoteToken` to `baseToken`.

Inputs

- `baseToken`
  - **Control**: Controlled by calling function.
  - **Constraints**: Checked to be different than zero and `quoteToken`.
  - **Impact**: The base token to swap from.
- `quoteAmount`
  - **Control**: Controlled by calling function.
  - **Constraints**: Checked to be above the amount of `quoteToken` that has to be swapped.
  - **Impact**: The amount of `quoteToken` to swap.
- `minBaseAmount`
  - **Control**: Controlled by calling function.
  - **Constraints**: Checked to be above the amount of `baseToken` that has to be received.
  - **Impact**: The minimum amount of `baseToken` to receive.
- `to`
  - **Control**: Controlled by calling function.
  - **Constraints**: Checked to be different than zero.
  - **Impact**: The address to send the swapped `baseToken2` to.

- `rebateTo`
  - **Control**: Controlled by calling function.
  - **Constraints**: None.
  - **Impact**: The address to send the rebate to.

Branches and code coverage

Intended branches

- Ensure that the spread changes after the swap. Currently not enforced.
  - □ Test coverage
- Ensure that `baseToken` is not zero or the `quoteToken`.
  - ☑ Test coverage
- Ensure that `to` is not zero.
  - ☑ Test coverage
- Ensure that the current balance minus the reserve is above `quoteAmount` (i.e., that tokens have been transferred beforehand).
  - ☑ Test coverage
- Should post the new price of `baseToken` after the swap on the `wooOracle`.
  - ☑ Test coverage
- Should calculate the swap fee and account for it in the `unclaimedFee`.
  - ☑ Test coverage
- Update the reserves of the quote and base tokens.
  - ☑ Test coverage
- Transfer the resulting `baseAmount` worth of `baseToken` to the `to` address.
  - ☑ Test coverage

Negative behavior

- Should not be callable when the contract is paused.
  - ☑ Negative test
- Should not reenter.
  - ☑ Negative test
- Should not allow swapping between the same tokens.
  - ☑ Negative test
- Should not allow performing the swap if no tokens have been transferred beforehand. Ensured in the `require` on balance check.
  - ☑ Negative test
- Should not allow a swap if `minBaseAmount` is not met.
  - ☑ Negative test

Function: `_swapBaseToBase(address baseToken1, address baseToken2, uint256 base1Amount, uint256 minBase2Amount, address to, address rebateTo)`

Allows swapping between two base tokens.

Inputs

- `baseToken1`
  - **Control**: Controlled by calling function.
  - **Constraints**: Checked to be different than zero and `quoteToken`.
  - **Impact**: The base token to swap from.
- `baseToken2`
  - **Control**: Controlled by calling function.
  - **Constraints**: Checked to be different than zero and `quoteToken`.
  - **Impact**: The base token to swap to.
- `base1Amount`
  - **Control**: Controlled by calling function.
  - **Constraints**: Checks that current balance is above that minus reserve.
  - **Impact**: The amount of `baseToken1` to swap.
- `minBase2Amount`
  - **Control**: Controlled by calling function.
  - **Constraints**: Checked to be above the amount of `baseToken2` that has to be received.
  - **Impact**: The minimum amount of `baseToken2` to receive.
- `to`
  - **Control**: Controlled by calling function.
  - **Constraints**: Checked to be different than zero.
  - **Impact**: The address to send the swapped `baseToken2` to.
- `rebateTo`
  - **Control**: Controlled by calling function.
  - **Constraints**: None.
  - **Impact**: The address to send the rebate to.

Branches and code coverage

Intended branches

- Ensure that the spread changes after each of the swaps. Currently not enforced.
  - ☐ Test coverage
- Ensure that `baseToken1` is not zero or the `quoteToken`.
  - ☑ Test coverage

- Ensure that `baseToken2` is not zero or the `quoteToken`.
  - ☑ Test coverage
- Ensure that `to` is not zero.
  - ☑ Test coverage
- Ensure that the current balance minus the reserve is above `base1Amount` (i.e., that tokens have been transferred beforehand).
  - ☑ Test coverage
- Should post the new price of `baseToken1` after the first swap on the `wooOracle`.
  - ☑ Test coverage
- Should calculate the swap fee and account for it in the `unclaimedFee`.
  - ☑ Test coverage
- Update the reserves of the quote and base tokens.
  - ☑ Test coverage
- Should post the new price of the `baseToken2` after the second swap on the `wooOracle`.
  - ☑ Test coverage
- Should transfer the resulting `base2Amount` worth of `baseToken2` to the `to` address.
  - ☑ Test coverage

Negative behavior

- Should not be callable when the contract is paused.
  - ☑ Negative test
- Should not re-enter.
  - ☑ Negative test
- Should not allow swapping between the same tokens.
  - ☑ Negative test
- Should not allow performing the swap if no tokens have been transferred beforehand. Ensured in the `require` on balance check.
  - ☑ Negative test
- Should not allow a swap if `minBase2Amount` is not met.
  - ☑ Negative test

5.3. Module: WooRouterV2.sol

Function: `externalSwap(address approveTarget, address swapTarget, address fromToken, address toToken, uint256 fromAmount, uint256 minToAmount, address payable to, bytes data)`

Performs a swap between two tokens using external (i.e., not WOOFI) `swapTarget`.

Inputs

- `approveTarget`
  - **Control**: Fully controlled by the caller.

    - **Constraints**: Has to be whitelisted.
    - **Impact**: The address to approve tokens for.
- `swapTarget`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: Has to be whitelisted.
    - **Impact**: The address that swaps the tokens.
- `fromToken`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: `msg.sender` needs to have enough balance.
    - **Impact**: The token to swap from.
- `toToken`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: `address(this)` needs to have enough balance.
    - **Impact**: The token to swap to.
- `fromAmount`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: `msg.sender` needs to have enough balance of `fromToken`.
    - **Impact**: The amount of `fromToken` to swap.
- `minToAmount`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: Checked that the returned `toToken`s are greater than or equal to this value.
    - **Impact**: The minimum amount of `toToken` to receive.
- `to`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: None. Checked that it is not the zero address.
    - **Impact**: The destination address to send the swapped tokens to.
- `data`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: None.
    - **Impact**: The data to pass to the `swapTarget`.

Branches and code coverage

Intended branches

- Ensure that `approveTarget` is not the zero address.
    ☑ Test coverage
- Ensure that `swapTarget` is not the zero address.
    ☑ Test coverage
- Ensure that `fromToken` is not the zero address.
    ☑ Test coverage

- Ensure that `toToken` is not the zero address.
  - ☑ Test coverage
- Ensure that `to` is not the zero address.
  - ☑ Test coverage
- Ensure that `approveTarget` is whitelisted.
  - ☑ Test coverage
- Ensure that `swapTarget` is whitelisted.
  - ☑ Test coverage
- If `fromToken` is not ETH, transfer `fromAmount` from the `msg.sender` and approve it to `approveTarget`.
  - ☑ Test coverage
- Call `swapTarget` with the provided `data` that facilitates the swap.
  - ☑ Test coverage
- Ensure the `swap` call was successful.
  - ☑ Test coverage
- Assure that the `post` balance of `toToken` is greater than or equal to the `pre` balance.
  - ☑ Test coverage
- Ensure that the `realToAmount` is greater than or equal to the `minToAmount` and greater than zero.
  - ☑ Test coverage
- Forward the recently swapped tokens to the `to` address.
  - ☑ Test coverage

Negative behavior

- Should not reenter.
  - ☑ Negative test

Function: `receive()`

Receive handler for calls with no calldata (transfers of native ETH).

Branches and code coverage

Intended branches

- Allows WETH address to send ETH to the contract.
  - ☑ Test coverage
- Allows whitelisted address to send ETH to the contract.
  - □ Test coverage

Negative behavior

- Reverts if an unknown sender sends ETH to the contract.
  - □ Negative test

Function: `swap(address fromToken, address toToken, uint256 fromAmount, uint256 minToAmount, address payable to, address rebateTo)`

Performs a swap between two tokens.

Inputs

- `fromToken`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: None. Checked that it is ETH or not.
    - **Impact**: The token to swap from.
- `toToken`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: None. Checked that it is ETH or not.
    - **Impact**: The token to swap to.
- `fromAmount`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: None. Checked that it is greater than zero.
    - **Impact**: The amount of `fromToken` to swap.
- `minToAmount`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: None. Checked that it is greater than zero.
    - **Impact**: The minimum amount of `toToken` to receive.
- `to`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: None. Checked that it is not the zero address.
    - **Impact**: The destination address to send the swapped tokens to.
- `rebateTo`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: None.
    - **Impact**: The address to send the rebate to.

Branches and code coverage

Intended branches

- Ensure that the tokens are not identical.
    - □ Test coverage
- Ensure `fromToken` is not the zero address.
    - ☑ Test coverage
- Ensure `toToken` is not the zero address.
    - ☑ Test coverage

- Ensure `to` is not the zero address.
  - ☑ Test coverage
- If `fromToken` is ETH, we assume the deposit is native; thus, `msg.value` should be greater than or equal to `fromAmount`. Deposit to `WETH` and transfer to `wooPool`.
  - ☑ Test coverage
- If `fromToken` is not ETH, transfer `fromAmount` from the sender to the `wooPool`.
  - ☑ Test coverage

Negative behavior

- Should not reenter.
  - ☑ Negative test

6. Assessment Results

At the time of our assessment, the reviewed code was partly deployed to the Arbitrum mainnet.

During our assessment on the scoped WooPool contracts, we discovered nine findings. No critical issues were found. Two findings were of high impact, three were of medium impact, two were of low impact, and the remaining findings were informational in nature.

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.