Zellic

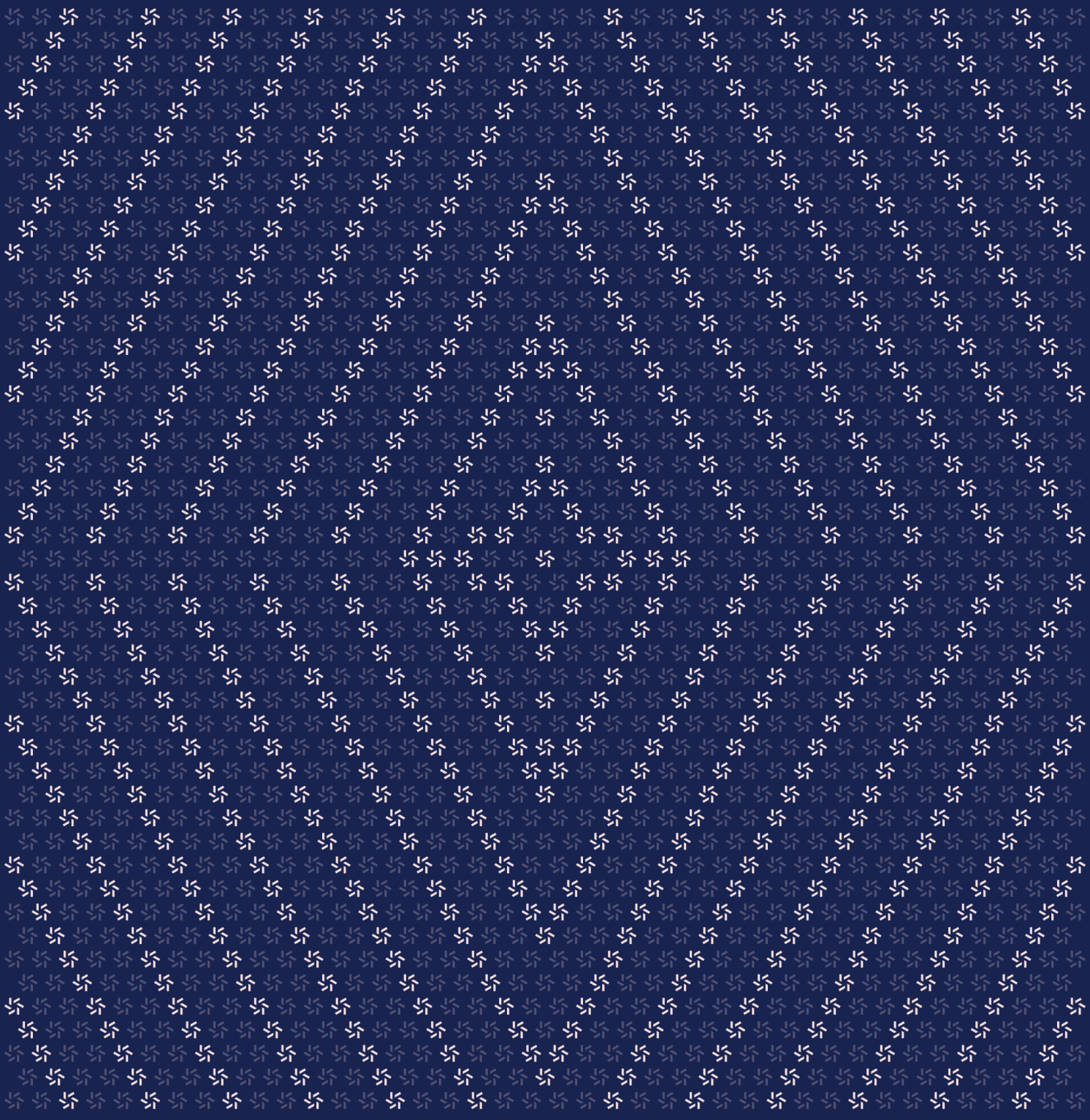
Prepared for
Chef Momo
PancakeSwap

Prepared by
Katerina Belotskaia
Daniel Lu
Zellic

October 1, 2024

PancakeSwap Infinity Periphery

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 PancakeSwap from July 9th to August 21st, 2024. During this engagement, Zellic reviewed PancakeSwap Infinity Periphery'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:

- Is the accounting done by the periphery contracts performed correctly?
- Are the contracts resilient to common attack vectors, such as reentrancy, overflow/underflow, and front-running that could lead to a loss of funds?

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

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

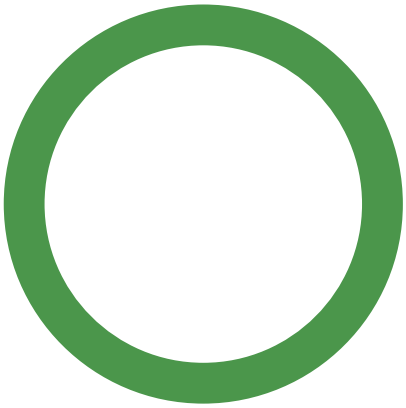
1.4. Results

During our assessment on the scoped PancakeSwap Infinity Periphery contracts, we discovered two findings, both of which were low impact.

Additionally, Zellic recorded its notes and observations from the assessment for PancakeSwap's benefit 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 PancakeSwap Infinity Periphery

PancakeSwap contributed the following description of PancakeSwap Infinity Periphery:

> PancakeSwap v4 is the latest iteration of the DEX, introducing several key features and improvements like Singleton contract, Flash Accounting, Hooks, and support for multiple AMM curves to enhance user experience and efficiency.

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:

PancakeSwap Infinity Periphery Contracts

Type	Solidity
Platform	EVM-compatible
Target	infinity-periphery
Repository	https://github.com/pancakeswap/infinity-periphery ↗
Version	`d3ccae2f4cbf319351157e6c6be667ea2d18a1e3`
Programs	`src/*`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of 7.05 person-weeks. The assessment was conducted by two consultants over the course of six calendar weeks.

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:

Katerina Belotskaia
Engineer
kate@zellic.io ↗

Daniel Lu
Engineer
daniel@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

July 9, 2024	Start of primary review period
July 10, 2024	Kick-off call
August 21, 2024	End of primary review period

3. Detailed Findings

3.1. Bin position manager may be incompatible with some hooks

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

Description

The BinPositionManager provides an interface for managing shares of pool bins as fungible tokens. In the `BIN_ADD_LIQUIDITY` and `BIN_REMOVE_LIQUIDITY` actions, the contract performs slippage checks.

```
BalanceDelta delta = binPoolManager.burn(
    params.poolKey,
    IBinPoolManager.BurnParams({ids: params.ids, amountsToBurn:
    params.amounts, salt: bytes32(0)}),
    ZERO_BYTES
);

// delta amt0/amt1 will either be 0 or positive in removing liquidity
if (delta.amount0() < 0 || delta.amount1() < 0) revert IncorrectOutputAmount();
if (uint128(delta.amount0()) < params.amount0Min || uint128(delta.amount1()) <
    params.amount1Min) {
    revert OutputAmountSlippage();
}
```

However, the assumption that the delta amounts must be positive in this case (or in the `_addLiquidity` case, negative) is not necessarily true. This is because hooks integrated with the particular pool may modify one or both of the amounts.

Impact

In some cases, this may result in the position manager being completely unable to withdraw deposited liquidity, locking user funds in the pool.

Recommendations

We recommend supporting cases where the sign of delta amounts is unexpected.

Remediation

This issue has been acknowledged by PancakeSwap, and a fix was implemented in commit `e76edc69`.

3.2. Bin position manager does not support hook data

Target	BinPositionManager		
Category	Business Logic	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

During modifications to liquidity with the BinPositionManager contract, the user is not able to provide data to `burn` and `mint` hooks. Instead, the hooks are unconditionally given `bytes(0)`.

```
BalanceDelta delta = binPoolManager.burn(
    params.poolKey,
    IBinPoolManager.BurnParams({ids: params.ids, amountsToBurn:
    params.amounts, salt: bytes32(0)}),
    ZERO_BYTES
);
```

This is in contrast to the CLPositionManager contract, which lets users provide this data when liquidity positions are modified.

```
// Can only call modify if there is non zero liquidity.
if (liquidity > 0) {
    (BalanceDelta liquidityDelta, BalanceDelta feesAccrued) =
        _modifyLiquidity(config, -(liquidity.toInt256()), bytes32(tokenId),
    hookData);
    // Slippage checks should be done on the principal liquidityDelta which is
    the liquidityDelta - feesAccrued
    (liquidityDelta - feesAccrued).validateMinOut(amount0Min, amount1Min);
}
```

Impact

Depending on the hook configuration of the particular pool, this may cause failures on the `burn` side and not the `mint` side. As a result, users may inadvertently lock their funds in the pool.

Recommendations

We recommend allowing the user to pass this data through. If the intention is specifically to disallow this, we would recommend preventing the use of the position manager on pools with such hooks.

Remediation

This issue has been acknowledged by PancakeSwap, and a fix was implemented in commit 435b6ca9 ↗.

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. Threat model

Position managers

The periphery contracts include BinPositionManager and CLPositionManager, which manage the positions of liquidity providers. Because they seek to provide a simpler interface by tokenizing positions (which may be transferred), the core protocol will see these contracts as the owners of the positions. This means that relative to other periphery contracts, their security is more critical.

Bin position manager

The BinPositionManager contract allows users to provide liquidity in exchange for fungible tokens. Because the underlying pool mechanics allow liquidity providers to target specific pools at specific prices, the contract provides logically separate tokens parameterized by the specific pool and price bin. These tokens correspond exactly shares of the bin in the core application.

CL position manager

Positions in concentrated liquidity pools are somewhat more complicated, because liquidity can be added at entire price ranges. The CLPositionManager gives a non-fungible token interface for managing these positions, where each token carries the data of which pool and price range it represents. To distinguish between separate tokens minted with the same data, the contract uses the token ID as a salt when interacting with the core protocol.

Just like in the bin position manager, these tokens are in one-to-one correspondence with positions in the underlying protocol.

Both the Bin and Cl position manager contracts provide functions for swaps with custom configurations for specific input and output token amounts, as well as support for both single and multiple swaps. After a single swap, the result amount is verified to ensure it is not less than the specified minimum expected output amount and not more than the expected input amount in the case of a swap with a specified output token. However, in the case of multiple swaps, this verification takes place only after all the swaps are completed, meaning the result of each individual swap is not checked. Additionally, the sign of the resulting delta after each swap will be indirectly verified during the call of the cast `toUint128` function, and in the case of an unexpected delta sign, the function will be reverted.

5. Assessment Results

At the time of our assessment, the reviewed code was not deployed to the Ethereum Mainnet.

During our assessment on the scoped PancakeSwap Infinity Periphery contracts, we discovered two findings, both of which were low impact.

5.1. 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.