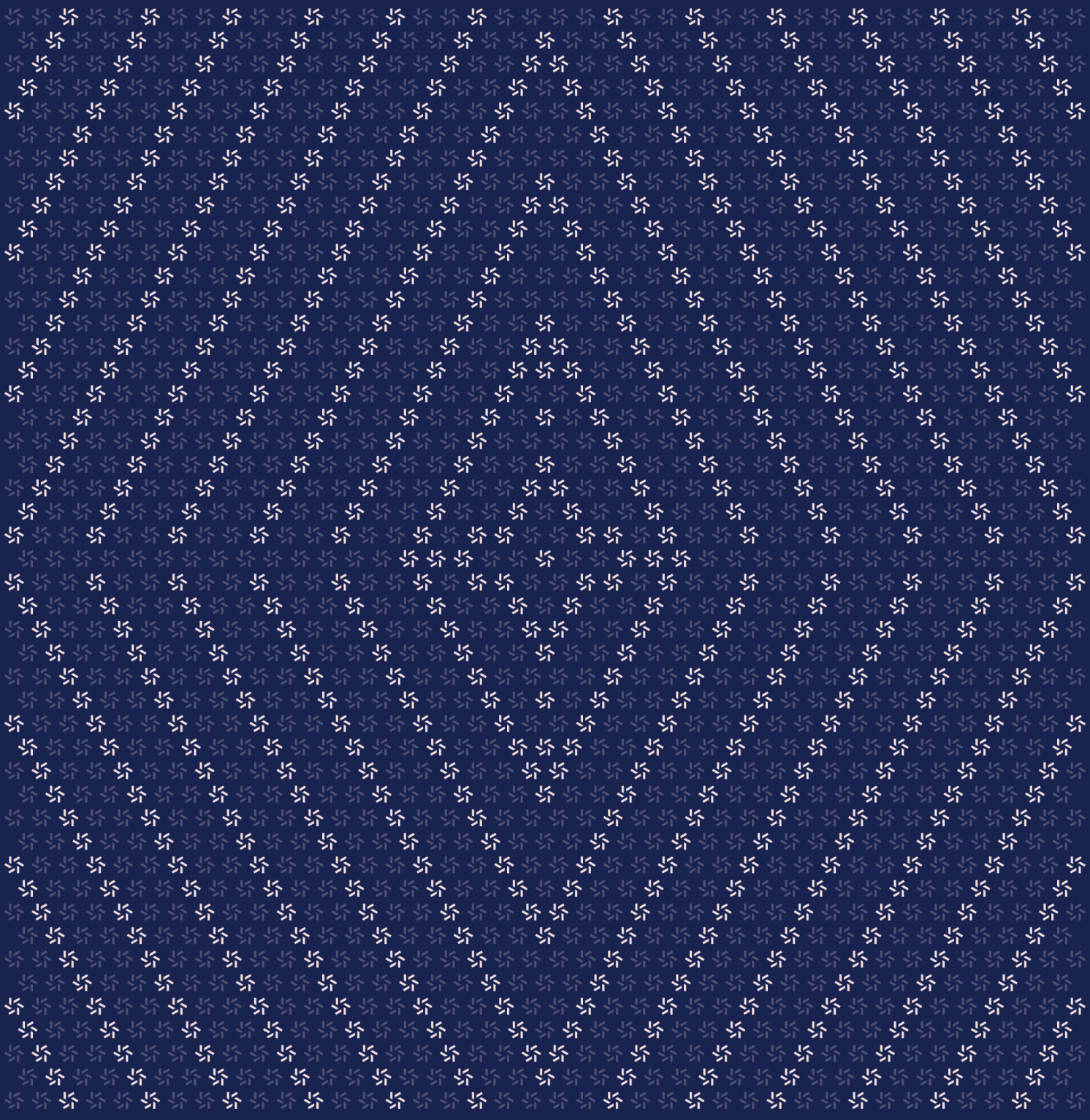
Zellic

Prepared for
Bob
SupSwap Labs

Prepared by
Junyi Wang
Zellic

February 6, 2024

SupSwap
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 SupSwap Labs on February 2nd, 2024. During this engagement, Zellic reviewed SupSwap's code for security vulnerabilities, design issues, and general weaknesses in security posture.

Zellic assumed that the original contracts forked by Mode Labs are secure. Only the changes made to the contracts by Mode Labs were in the scope of this review.

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 question:

- Do the changes to the base contracts introduce any new vulnerabilities?

1.3. Non-goals and Limitations

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

- The fee-sharing contract
- Unchanged parts of the fork from Uniswap
- Unchanged parts of the fork from PancakeSwap
- Unchanged parts of code forked from other places
- Correctness of hardcoded address constants

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

We assessed changes only and have not assessed any code forked from elsewhere.

1.4. Results

During our assessment on the scoped SupSwap contracts, we discovered one finding, which was informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for SupSwap Labs'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	0
Informational	1

2. Introduction

2.1. About SupSwap

SupSwap is a cost efficient liquidity layer on Mode Network.

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:

SupSwap Contracts

Repository	https://github.com/SupSwap/supswap-contracts-audit ↗
Version	supswap-contracts-audit: `b7f5509cfe5ea5183287ebdbc30b43ea968bc731`
Program	projects/*.sol
Type	Solidity
Platform	EVM-compatible

2.4. Project Overview

Zellic was contracted to perform a security assessment with one consultant for a total of one person-day. The assessment was conducted over the course of one person-day.

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:

Junyi Wang
Engineer
junyi@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

February 2, 2024	Start of primary review period
February 2, 2024	End of primary review period

3. Detailed Findings

3.1. Incorrectly documented fee

Target	v2-core/SupRouter		
Category	Code Maturity	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The swap fee to liquidity providers documented in the comment is incorrect. See the below code.

```
// if fee is on, mint liquidity equivalent to 1/6th of the growth in sqrt(k)
function _mintFee(uint112 _reserve0, uint112 _reserve1)
    private
    returns (bool feeOn)
{
    address feeTo = ISupFactory(factory).feeTo();
    feeOn = feeTo != address(0);
    uint256 _kLast = kLast; // gas savings
    if (feeOn) {
        if (_kLast != 0) {
            uint256 rootK = Math.sqrt(uint256(_reserve0).mul(_reserve1));
            uint256 rootKLast = Math.sqrt(_kLast);
            if (rootK > rootKLast) {
                uint256 numerator = totalSupply.mul(rootK.sub(rootKLast));
                uint256 denominator = rootK.mul(3).add(rootKLast);
                uint256 liquidity = numerator / denominator;
                if (liquidity > 0) _mint(feeTo, liquidity);
            }
        }
    } else if (_kLast != 0) {
        kLast = 0;
    }
}
```

The fee is documented as 1/6, but the actual fee is 1/4.

Recommendations

Correct the comment.

Remediation

This issue has been acknowledged by SupSwap Labs.

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. Summary of changes

User fees to LPs

The user fee to LPs was changed from 25 basis points to 20 basis points.

Protocol tax on LP gain

The protocol fee was changed from 8/25 to 1/4.

Removed interface method

The following function was removed from the interface of the v2 Factory.

```
function INIT_CODE_PAIR_HASH() external view returns (bytes32);
```

Constructor additions

The following code was added to some of the constructors.

```
IFeeSharing feeSharing
    = IFeeSharing(0x8680CEaBcb9b56913c519c069Add6Bc3494B7020); // This address
    is the address of the SFS contract
feeSharing.assign(82); //Registers this contract and assigns the NFT to the
    owner of this contract
```

5. Assessment Results

At the time of our assessment, the reviewed code was deployed to the Mode Network.

During our assessment on the scoped SupSwap contracts, we discovered one finding, which was informational in nature. SupSwap Labs acknowledged the finding and implemented a fix.

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.