Zellic

Trident Concentrated Liquidity

Smart Contract Security Assessment

January 31, 2023

Prepared for:

SushiSwap

Prepared by:

Katerina Belotskaia and Vlad Toie

Zellic Inc.

Contents

About Zellic

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

Zellic aims to treat clients on a case-by-case basis and to consider their individual, unique concerns and business needs. Our goal is to see the long-term success of our partners rather than simply provide a list of present security issues. Similarly, we strive to adapt to our partners' timelines and to be as available as possible. To keep up with our latest endeavors and research, check out our website zellic.io or follow @zellic_io on Twitter. If you are interested in partnering with Zellic, please contact us at hello@zellic.io.

1 Executive Summary

Zellic conducted a security assessment for SushiSwap from January 25th to January 27th, 2023. During this engagement, Zellic reviewed Trident Concentrated Liquidity's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.1 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:

- Could an attacker drain liquidity from the BentoBox?
- Do the implemented changes to UniV3 disrupt the liquidity pool in any way?
- Could an attacker leverage the BentoBox's functionality to their advantage?
- What implications does the new BentoBox system have for the protocol's architecture?

1.2 Non-Goals and Limitations

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

- Problems relating to the front-end components and infrastructure of the project.
- Problems due to improper key custody or off-chain access control.
- Issues stemming from code or infrastructure outside of the assessment scope.

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide. Moreover, it is important to mention that Zellic was asked to perform a review over changes between two commits, 05c10bf6 and 09653099, and not a review of the entire codebase. For that reason, we have made assumptions in terms of the inherent security of the contracts and performed our review based on them.

1.3 Results

During our assessment on the scoped Trident Concentrated Liquidity contracts, we did not discover any findings. In the discussion section, we have elaborated on our notes throughout the security review as well as future considerations for the contracts.

Zellic recorded its notes and observations from the assessment for SushiSwap's benefit in the Discussion section (3) at the end of the document.

2 Introduction

2.1 About Trident Concentrated Liquidity

Trident Concentrated Liquidity is an update of the Uniswap V3 Protocol to support BentoBox.

Sushi is a community-driven organization built to solve what might be called the "liquidity problem." One could define this problem as the inability of disparate forms of liquidity to connect with markets in a decentralized way, and vice versa. While other solutions provide incrementally progressive advances toward solving the problem of liquidity, Sushi's progress is intended to create a broader range of network effects. Rather than limiting itself to a single solution, Sushi intertwines many decentralized markets and instruments.

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 the

contracts' 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 code base in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradeability weaknesses, centralization risks, and so 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.

2.3 Scope

The engagement involved a review of the following targets:

Trident Concentrated Liquidity Contracts

Repository	https://github.com/sushiswap/trident-cl
Versions	`0965309999e755a8fca8c28ee98c6058b54e70c0`
Programs	• TridentCLPool.sol • TridentCLFactory.sol • TridentCLPoolDeployer.sol • ITridentCLFactory.sol • ITridentCLPool.sol • ITridentCLPoolDeployer.sol • ITridentCLPoolActions.sol • ITridentCLPoolDerivedState.sol • ITridentCLPoolEvents.sol • ITridentCLPoolImmutables.sol • ITridentCLPoolOwnerActions.sol • ITridentCLPoolState.sol • ITridentCLFlashCallback.sol • ITridentCLMintCallback.sol • ITridentCLSwapCallback.sol
Type	Solidity
Platform	EVM-compatible

2.4 Project Overview

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

Contact Information

The following project managers were 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

Vlad Toie, Engineer
vlad@zellic.io

2.5 Project Timeline

The key dates of the engagement are detailed below.

January 25, 2023	Start of primary review period
January 27, 2023	End of primary review period

3 Discussion

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

As mentioned in the executive summary, the review was focused on the changes made in a specific point of the codebase, in two particular commits, 05c10bf6 and 09653099, rather than an overall examination of the entire codebase. With this in mind, certain assumptions were made about the security of the contracts, and the review was conducted accordingly.

The most significant change over the default `UniswapV3Pool` and associate contracts, such as the Factory and Deployer, is the introduction of the `BentoBox` as a means of storing the tokens on behalf of the pool. The implications of this have been described below as part of the security review process.

3.1 Changes review

The `TridentCLPool`, `TridentCLFactory`, and `TridentCLPoolDeployer` contracts have been updated.

- **Updated the `deploy` function of the `TridentCLPoolDeployer` contract.** The `bentoBox` argument was added and passed in the `parameters` struct to be used in the liquidity pool.
- **Updated the constructor of the `TridentCLFactory` contract.** Added the `_bentoBox` argument. This address is used in the `createPool` function to deploy a new liquidity pool.

The changes listed below relate to the **`TridentCLPool`** contract:

- **The BentoBox usage.** The most important change here is the use of BentoBox, which stores tokens on behalf of the pool contract, like a vault, and has additional functionalities of its own.
- **Added the internal `_transfer` function.** This function allows direct BentoBox transfers or withdrawals towards the `to` address. Previously, it was a `safeTransfer` of the specific token towards `to`. It assumes that `token0` and `token1` can be both withdrawn and transferred from BentoBox. It is used in `collectProtocol()`, `collect()`, `swap()`, and `flash()`.
- **Updated the `balance0()` and the `balance1()` functions.** The changes relate to the fact that BentoBox is now the vault for the `token0` and the `token1`. It is important

to note that neither `balance0` nor `balance1` are updated if a direct ERC20 transfer happens to the pool. Therefore, all token transfers must occur through the BentoBox functions.

- **Updated the `collect()`, `swap()`, `flash()`, and `collectProtocol()` functions.** The `unwrapBento` flag was added. This flag is used in the `_transfer` function and must determine whether to transfer or to withdraw (e.g., unwrap and then transfer as native) tokens from the vault.
- **Added new structure `FlashCache`.** This structure is used in the `flash` function to pack the data set in memory.
- **Updated the `SwapCache` structure.** Added new flags `zeroForOne` and `unwrapBento`. This structure is used in the `swap` function. The flag `zeroForOne` was already used there but now is grouped with other data into this structure. The `unwrapBento` flag is necessary for the `_transfer` function to determine whether to withdraw tokens from the vault or transfer after the swap.

3.2 Possible fees difference between `flash` and `flashLoan`

As per the current implementation, the `BentoBox` has been integrated in the `TridentCLPool`. Therefore, the inherent `flashLoan` from `BentoBox` will be available to users, on top of the previously existing `flash` from `TridentCLPool`. Technically the amount of tokens that can be loaned is different between the two contracts. The main reason for that is because `BentoBox` should handle a greater amount of liquidity than the `TridentCLPool`, since by design the tokens that `TridentCLPool` owns are actually stored within the `BentoBox`, and they can therefore be loaned as part of the `BentoBox` loan functionality.

Considering the possibility of performing different flashloans from presumably two interconnected contracts, it is important that the fees that each of these loans incur are similar, such that there exists no economic incentive to choose one contract over the other, apart from the need for higher levels of liquidity.

4 Audit Results

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

During our audit, we did not discover any findings.

4.1 Disclaimers

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

For each finding, Zellic provides a recommended solution. All code in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code.

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