Zellic

Prepared for
Daniel Alcarraz
GammaSwap Protocol

Prepared by
Filippo Cremonese
Sina Pilehchiha
Zellic

November 7, 2023

Deltaswap

Smart Contract Security Assessment

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 GammaSwap Protocol from October 31st to November 1st, 2023. During this engagement, Zellic reviewed Deltaswap'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:

- Is the tracking of the average volume correct?
- Is the trading fee being charged when intended by the protocol design?
- Are there DOS risks due to the tracking of the average trading volume?

1.2. 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.

1.3. Results

During our assessment on the scoped Deltaswap contracts, we discovered one finding, which was of high impact.

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

Breakdown of Finding Impacts

Impact Level	**Count**
🟥 Critical	0
🟧 High	1
🟨 Medium	0
🟩 Low	0
⬜ Informational	0

2. Introduction

2.1. About Deltaswap

Deltaswap is a constant-function market maker that uses the constant product formula $(x * y = k)$ based on a fork of Uniswap V2. Unlike Uniswap V2, there are no fees if the size of the transaction is below a certain percentage of the average liquidity in the pool.

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:

Deltaswap Contracts

Repository	https://github.com/gammaswap/v1-deltaswap ↗
Version	v1-deltaswap: `2df679b008d4a6eca8f44e87cd3ec7bd7c7bc581`
Programs	• DeltaSwapPair • DeltaSwapERC20 • DeltaSwapRouter01 • DeltaSwapRouter02 • DeltaSwapFactory
Type	Solidity
Platform	EVM-compatible

2.4. Project Overview

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

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:

Filippo Cremonese
Engineer
fcremo@zellic.io ↗

Sina Pilehchiha
Engineer
sina@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

September 1, 2023	Kick-off call
September 1, 2023	Start of primary review period
September 7, 2023	End of primary review period

3. Detailed Findings

3.1. Incorrect trade-volume calculation

Target	DeltaSwapPair		
Category	Business Logic	Severity	High
Likelihood	High	Impact	High

Description

DeltaSwap is intended to charge swap fees only if the recent trade volume is greater than a configurable percentage of the recent average liquidity. The two averages are computed as exponential moving averages (EMAs), giving more weight to more recent data, based on the difference between the block number when the EMA was last updated and the current block number.

The `swap` function determines the fee to be applied (in basis points) with the following line of code (reformatted for readability)

```
fee = calcTradingFee(
    _updateLiquidityTradedEMA(
        DSMath.calcTradeLiquidity(amount0In, amount1In, _reserve0,
    _reserve1)
    ),
    liquidityEMA
)
```

The line also computes and updates the trade liquidity EMA value, which is then passed to `calcTradingFee` to compute the trading fee.

The problem lies in how `calcTradeLiquidity` determines the trade liquidity amount, since it is assuming only one of `amount0In` and `amount1In` are greater than zero:

```
function calcTradeLiquidity(uint256 amount0, uint256 amount1,
    uint256 reserve0, uint256 reserve1) internal pure returns(uint256) {
    if(amount0 > 0) {
        return calcSingleSideLiquidity(amount0, reserve0, reserve1);
    } else if(amount1 > 0) {
        return calcSingleSideLiquidity(amount1, reserve1, reserve0);
    }
    return 0;
}
```

```
function calcSingleSideLiquidity(uint256 amount, uint256 reserve0,
    uint256 reserve1) internal pure returns(uint256) {
    uint256 amount0 = amount / 2;
    uint256 amount1 = amount0 * reserve1 / reserve0;
    return DSMath.sqrt(amount0 * amount1);
}
```

Impact

A user could leverage this behavior to artificially lower the traded volume when trading `token1` for `token0`. To do so, they can perform a swap, making sure to supply an `amount0Out` value that is slightly less than the maximum value that would still satisfy the K-invariant. Alternatively, they could also use the callback to transfer a small amount of `token0` to the contract.

This would cause `calcTradeLiquidity` to consider the small amount of input `token0` to calculate the trade volume, returning an artificially low trade volume and also influencing the trade EMA less than intended.

Recommendations

Consider updating function `DeltaSwapPair.swap()` to charge a trading fee that depends on `max(amount0, amount1)` instead of only one of the values being greater than zero in function `DSMath.calcTradeLiquidity()`.

Remediation

This issue has been acknowledged by GammaSwap Protocol, and a fix was implemented in commit 07e3464c ↗..

The `calcTradeLiquidity` function was changed to return the maximum trade of the two individually computed single sided trade liquidities. In addition, `calcTradingFee` was updated to consider the greater number between the liquidity EMA and the current trade liquidity when determining whether fees should be charged.

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. Uniswap differences

The code is forked from Uniswap V2 and maintains a strong similarity. Apart from renaming variables, the following changes can be observed in the DeltaSwapPair contract when comparing DeltaSwap with Uniswap V2 at commit ee547b17 ↗:

- The swap fee is adjustable instead of a fixed three base points, as it is obtained by calling the factory.
- The `uint` was replaced with the more explicit `uint256`.
- The target Solidity version was bumped, and `SafeMath` operations were replaced by built-in checked math.

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: DeltaSwapPair.sol

Function: `burn(address to)`

This function can be called to burn liquidity tokens in exchange for a corresponding share of underlying assets.

Inputs

- `to`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Recipient of the assets.

Branches and code coverage

Intended branches

- Burns liquidity tokens and transfers the corresponding amount of underlying assets to the recipient.
    - ☑ Test coverage

Negative behavior

- Reverts if zero assets are transferred.
    - ☑ Negative test

Function call analysis

- `IERC20(_token0).balanceOf(address(this))`
    - **What is controllable?** None.
    - **If the return value is controllable, how is it used and how can it go wrong?** Return value can be increased by transferring `asset0` to the contract, but it is not useful for an attacker.
    - **What happens if it reverts, reenters or does other unusual control flow?** Reverts are propagated upwards; reentrancy is prevented via

reentrancy guard.

- `IERC20(_token1).balanceOf(address(this))`
    - **What is controllable?** None.
    - **If the return value is controllable, how is it used and how can it go wrong?** Return value can be increased by transferring `asset1` to the contract, but it is not useful for an attacker.
    - **What happens if it reverts, reenters or does other unusual control flow?** Reverts are propagated upwards; reentrancy is prevented via reentrancy guard.
- `this._mintFee(_reserve0, _reserve1) -> IDeltaSwapFactory(this.factory).feeTo()`
    - **What is controllable?** Nothing.
    - **If the return value is controllable, how is it used and how can it go wrong?** Not controllable.
    - **What happens if it reverts, reenters or does other unusual control flow?** Cannot revert nor reenter.
- `IERC20(_token0).balanceOf(address(this))`
    - **What is controllable?** None.
    - **If the return value is controllable, how is it used and how can it go wrong?** Return value cannot be controlled independently from the previous `balanceOf` call.
    - **What happens if it reverts, reenters or does other unusual control flow?** Reverts are propagated upwards; reentrancy is prevented via reentrancy guard.
- `IERC20(_token1).balanceOf(address(this))`
    - **What is controllable?** None.
    - **If the return value is controllable, how is it used and how can it go wrong?** Return value cannot be controlled independently from the previous `balanceOf` call.
    - **What happens if it reverts, reenters or does other unusual control flow?** Reverts are propagated upwards; reentrancy is prevented via reentrancy guard.

Function: `mint(address to)`

This function can be called to mint liquidity tokens in exchange for providing liquidity.

Inputs

- `to`
    - **Control**: Arbitrary.
    - **Constraints**: None.
    - **Impact**: Recipient of the tokens.

Branches and code coverage

Intended branches

- Computes and mints the amount of liquidity tokens owed to the caller (initial case when `_totalSupply` is zero).
  - ✓ Test coverage
- Computes and mints the amount of liquidity tokens owed to the caller (`_totalSupply` is not zero).
  - ✓ Test coverage
- Handles minting fees.
  - ✓ Test coverage
- Updates cached reserves.
  - ✓ Test coverage
- Updates cached liquidity EMA value every time a new block is processed.
  - ✓ Test coverage

Negative behavior

- Reverts if no liquidity tokens are minted.
  - ✓ Negative test

Function call analysis

- `IERC20(this.token0).balanceOf(address(this))`
  - **What is controllable?** Nothing.
  - **If the return value is controllable, how is it used and how can it go wrong?** Used to determine the amount of `token0` in.
  - **What happens if it reverts, reenters or does other unusual control flow?** Reverts are propagated upwards. Reentrancy is prevented via reentrancy guards.
- `IERC20(this.token1).balanceOf(address(this))`
  - **What is controllable?** Nothing.
  - **If the return value is controllable, how is it used and how can it go wrong?** Used to determine the amount of `token1` in.
  - **What happens if it reverts, reenters or does other unusual control flow?** Reverts are propagated upwards. Reentrancy is prevented via reentrancy guards.
- `this._mintFee(_reserve0, _reserve1) -> IDeltaSwapFactory(this.factory).feeTo()`
  - **What is controllable?** Nothing.
  - **If the return value is controllable, how is it used and how can it go wrong?** Not controllable.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.

- `this._update(balance0, balance1, _reserve0, _reserve1)`
  - **What is controllable?** `balance0` and `balance1`, by transferring to the contract.
  - **If the return value is controllable, how is it used and how can it go wrong?** Not used.
  - **What happens if it reverts, reenters or does other unusual control flow?** Reverts are propagated upwards. Reentrancy is not possible.

Function: `swap(uint256 amount0Out, uint256 amount1Out, address to, byte[] data)`

This function can be called to perform a swap. It also supports flash swaps.

Inputs

- `amount0Out`
  - **Control**: Arbitrary.
  - **Constraints**: `amount0Out < _reserve0`, `amount0Out > 0 || amount1Out > 0`.
  - **Impact**: Amount to get from the swap.
- `amount1Out`
  - **Control**: Arbitrary.
  - **Constraints**: `amount1Out < _reserve1`, `amount0Out > 0 || amount1Out > 0`.
  - **Impact**: Amount to get from the swap.
- `to`
  - **Control**: Arbitrary.
  - **Constraints**: `to != _token0 && to != _token1`.
  - **Impact**: Recipient of the output tokens and (optionally) implementer of the `deltaSwapCall` callback.
- `data`
  - **Control**: Arbitrary.
  - **Constraints**: None.
  - **Impact**: If not empty, a callback is made to the recipient with this data after the output assets are transferred — typically used to implement flash loans.

Branches and code coverage

Intended branches

- Optimistically sends the requested output to the recipient, invokes the recipient

callback if required, computes input amounts, updates traded liquidity EMA, ensures K-invariant still holds (accounting for fees), and updates reserves.
    - ☑ Test coverage
- Handles fee calculation if caller is GammaSwap pool.
    - ☑ Test coverage
- Handles fee calculation if caller is not GammaSwap pool.
    - ☑ Test coverage

Negative behavior

- Reverts if both `amount0Out` and `amount1Out` are zero.
    - ☑ Negative test
- Reverts if `amount0Out >= _reserve0`.
    - ☑ Negative test
- Reverts if `amount1Out >= _reserve1`.
    - ☑ Negative test
- Reverts if `to == _token0 || to == _token1`.
    - ☑ Negative test
- Reverts if the callback reverts.
    - ☑ Negative test
- Reverts if `amount0In == 0 && amount1In == 0`.
    - ☑ Negative test
- Reverts if K-invariant does not hold at the end of the transaction.
    - ☑ Negative test

Function call analysis

- `IDeltaSwapCallee(to).deltaSwapCall(msg.sender, amount0Out, amount1Out, data)`
    - **What is controllable?** `amount0Out`, `amount1Out`, and `data`.
    - **If the return value is controllable, how is it used and how can it go wrong?** Not used.
    - **What happens if it reverts, reenters or does other unusual control flow?** Reverts are propagated upwards; reentrancy is prevented by reentrancy guards.
- `IERC20(_token0).balanceOf(address(this))`
    - **What is controllable?** Nothing.
    - **If the return value is controllable, how is it used and how can it go wrong?** Controllable by transferring assets to the contract — used to determine `token0` input amount.
    - **What happens if it reverts, reenters or does other unusual control flow?** Reverts are propagated upwards; reentrancy is prevented by reentrancy guards.
- `IERC20(_token1).balanceOf(address(this))`

    - **What is controllable?** Nothing.
    - **If the return value is controllable, how is it used and how can it go wrong?** Controllable by transferring assets to the contract — used to determine `token1` input amount.
    - **What happens if it reverts, reenters or does other unusual control flow?** Reverts are propagated upwards; reentrancy is prevented by reentrancy guards.
- `DSMath.calcTradeLiquidity(amount0In, amount1In, _reserve0, _reserve1)`
    - **What is controllable?** `amount0In` and `amount1In`.
    - **If the return value is controllable, how is it used and how can it go wrong?** It is influenced by `amount0In` and `amount1In`. Used as the trade volume.
    - **What happens if it reverts, reenters or does other unusual control flow?** Cannot revert nor reenter.
- `this._updateLiquidityTradedEMA(...)`
    - **What is controllable?** The first argument is the return value of the call above.
    - **If the return value is controllable, how is it used and how can it go wrong?** It is not directly controlled but influenced by the argument, which is used to update the EMA.
    - **What happens if it reverts, reenters or does other unusual control flow?** Cannot revert nor reenter.
- `this.calcTradingFee(...)`
    - **What is controllable?** The first argument is the return value of the call above.
    - **If the return value is controllable, how is it used and how can it go wrong?** Not directly controllable but influenced by the EMA.
    - **What happens if it reverts, reenters or does other unusual control flow?** Cannot revert nor reenter (only external call is made to the factory).
- `IDeltaSwapFactory(this.factory).gsFee()`
    - **What is controllable?** Nothing.
    - **If the return value is controllable, how is it used and how can it go wrong?** Not controllable.
    - **What happens if it reverts, reenters or does other unusual control flow?** Cannot revert; if it did, it would be propagated upward. Reentrancy is not possible due to reentrancy guards.

6. Assessment Results

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

During our assessment on the scoped Deltaswap contracts, we discovered one finding, which was of high impact. GammaSwap Protocol acknowledged the finding and implemented a fix.

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.