Zellic

GammaSwap

Smart Contract Security Assessment

March 15, 2023

Prepared for:

Daniel Alcarraz

GammaSwap

Prepared by:

Ulrich Myhre and Katerina Belotskaia

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 GammaSwap from March 13th to March 14th, 2023. During this engagement, Zellic reviewed GammaSwap'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:

- Are pool funds (in terms of liquidity invariant units) unchanged or increased after executing flash loans?
- And if it is unchanged, did the liquidity invariant debt of a loan increase?
- Can a flash loan that leads to collateral rebalancing violate the collateral balances of other users?
- Can GammaPool be attacked or produce undesired results when using a flash loan, especially in interaction with CFMMs?
- Are there risks for miscalculations or rounding errors that lead to loss of funds, especially in the liquidation strategy?
- Is any of the overridden functionality-changing behavior in concrete, existing strategies?

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
- Code outside the scope of the engagement

1.3 Results

During our assessment on the scoped GammaSwap contracts, we discovered one finding. No critical issues were found. The finding was informational in nature.

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	0
Low	0
Informational	1

2 Introduction

2.1 About GammaSwap

GammaSwap is the modular scaling layer for DeFi liquidity, offering leveraged exposure to any token by turning impermanent loss into impermanent gains. It integrates existing AMM architectures and allows users to borrow LP tokens to take leveraged exposure to the volatility of the underlying assets.

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:

GammaSwap Contracts

Repositories	https://github.com/gammaswap/v1-core https://github.com/gammaswap/v1-implementations
Versions	v1-core: `17fd0c9754095a6b05e7f1be335f1d4262e49507` v1-implementations: `d308cffac63e872b793f0c9c3e897a1798bf7601`
Programs	• ExternalLongStrategy • ExternalLiquidationStrategy • ExternalBaseStrategy • BalancerExternalLiquidationStrategy • BalancerExternalLongStrategy • CPMMExternalLiquidationStrategy • CPMMExternalLongStrategy
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:

Ulrich Myhre, Engineer
unblvr@zellic.io

Katerina Belotskaia, Engineer
kate@zellic.io

2.5 Project Timeline

The key dates of the engagement are detailed below.

March 13, 2023	Start of primary review period
March 14, 2023	End of primary review period

3 Detailed Findings

3.1 Lack of _loan validation

- **Target**: ExternalLiquidationStrategy, ExternalLongStrategy
- **Category**: Coding Mistakes
- **Likelihood**: Low
- **Severity**: Informational
- **Impact**: Informational

Description

There is a lack of validation on the `_loan` object.

1. The `ExternalLongStrategy._rebalanceExternally` function calls `_getLoan` in order to retrieve the `_loan` associated with the specified `tokenId` and verify that the `msg.sender` is the creator of the loan.

```
function _getLoan(uint256 tokenId)
    internal virtual view returns(LibStorage.Loan storage _loan) {
     _loan = s.loans[tokenId];
     if(tokenId ≠ uint256(keccak256(abi.encode(msg.sender,
    address(this), _loan.id)))) {
          revert Forbidden();
     }
 }
```

However, there is currently no check in place to ensure that the `_loan` exists and that the `_loan.id` is not a zero value.

Therefore, to bypass the check that the `msg.sender` is the loan's creator, a caller can pass the `tokenId` equal to `uint256(keccak256(abi.encode(msg.sender, address(this), _loan.id))` where `_loan.id` equals zero.

This would cause the function to return an empty `_loan` object.

2. There is not a check that the `_loan` associated with `tokenId` exists inside the `ExternalLiquidationStrategy._liquidateExternally` function.

```
function _liquidateExternally(uint256 tokenId, uint128[]
    calldata amounts, uint256 lpTokens, address to, bytes calldata data)
    external override lock virtual returns(uint256 loanLiquidity,
    uint256[] memory refund) {
     uint128[] memory tokensHeld;
     uint256 writeDownAmt;
     uint256 collateral;

     LibStorage.Loan storage _loan = s.loans[tokenId];

     (loanLiquidity, collateral, tokensHeld, writeDownAmt)
    = getLoanLiquidityAndCollateral(_loan, s.cfmm);
  ...
}
```

As a result, the empty `_loan` object can be used inside this function.

Impact

Calling these functions with an empty `_loan` object will result in a reversion with an out-of-bounds error when trying to iterate over the empty `tokensHeld` array stored within the `_loan` object. However, even though this error provides a measure of protection, it is still crucial to perform explicit verification of data to prevent unexpected behavior during function execution.

Recommendations

Add a check that the `_loan` associated with `tokenId` is not empty.

```
function _getLoan(uint256 tokenId)
    internal virtual view returns(LibStorage.Loan storage _loan) {
     _loan = s.loans[tokenId];
    require(_loan.id ≠ 0)
     if(tokenId ≠ uint256(keccak256(abi.encode(msg.sender,
    address(this), _loan.id)))) {
         revert Forbidden();
     }
 }
```

```
function _liquidateExternally(uint256 tokenId, uint128[]
    calldata amounts, uint256 lpTokens, address to, bytes calldata data)
    external override lock virtual returns(uint256 loanLiquidity,
    uint256[] memory refund) {
     uint128[] memory tokensHeld;
     uint256 writeDownAmt;
     uint256 collateral;

     LibStorage.Loan storage _loan = s.loans[tokenId];
     require(_loan.id != 0)
     (loanLiquidity, collateral, tokensHeld, writeDownAmt)
    = getLoanLiquidityAndCollateral(_loan, s.cfmm);
  ...
}
```

Remediation

GammaSwap acknowledged this finding and implemented a fix in commit ec1a429e.

4 Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the smart 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.

4.1 Module: ExternalLiquidationStrategy.sol

Function: `_liquidateExternally(uint256 tokenId, uint128[] amounts, uint256 lpTokens, address to, byte[] data)`

Allows any caller to liquidate the existing loan using a flash loan of collateral tokens from the pool and/or CFMM LP tokens. Before the liquidation, the `externalSwap` function will be called. After that, a check will be made that enough tokens have been deposited. Allows only full liquidation of the loan.

Inputs

- `tokenId`
  - **Validation**: There is no verification that the corresponding `_loan` for this `tokenId` exists.
  - **Impact**: A `tokenId` referring to an existing `_loan`. Not necessary `msg.sender` is owner of `_loan`, so the caller can choose any existing loan.
- `amounts`
  - **Validation**: There is a check that `amount <= s.TOKEN_BALANCE` inside `externalSwap`→`sendAndCalcCollateralLPTokens`→`sendToken` function.
  - **Impact**: Amount of tokens from the pool to flash loan.
- `lpTokens`
  - **Validation**: There is a check that `lpTokens <= s.LP_TOKEN_BALANCE` inside `externalSwap`→`sendCFMMLPTokens`→`sendToken` function
  - **Impact**: Amount of CFMM LP tokens being flash loaned.
- `to`
  - **Validation**: Cannot be zero address.
  - **Impact**: Address that will receive the collateral tokens and/or `lpTokens` in flash loan.

- `data`
  - **Validation**: No checks.
  - **Impact**: Custom user data. It is passed to the `externalCall`.

Branches and code coverage (including function calls)

The part of `_liquidateExternally` tests are skipped.

Intended branches

- ☐ Check that loan was fully liquidated

Negative behavior

- ☑ `_loan` for `tokenId` does not exist.
- ☐ Balance of contract not enough to transfer `amounts`.
- ☐ Balance of contract not enough to transfer `lpTokens`.
- ☐ Zero `to` address.
- ☑ After `externalCall` the `s.cfmm` balance of contract has not returned to the previous value.
- ☐ After `externalCall` the balance of contract for each `tokens` has not returned to the previous value.

Function call analysis

- `externalSwap(_loan, s.cfmm, amounts, lpTokens, to, data) → sendAndCalcCollateralLPTokens(to, amounts, lastCFMMTotalSupply) → sendToken(IERC20(tokens[i]), to, amounts[i], s.TOKEN_BALANCE[i], type(uint128).max) → GammaSwapLibrary.safeTransfer(token, to, amount)`
  - **External/Internal?** External.
  - **Argument control?** `to` and `amount`.
  - **Impact**: The caller can transfer any number of tokens that is less than `s.TOKEN_BALANCE[i]`, but they must return the same or a larger amount after the `externalCall` function call; it will be checked inside the `updateCollateral` function.
- `externalSwap(_loan, s.cfmm, amounts, lpTokens, to, data) → sendCFMMLPTokens(_cfmm, to, lpTokens) → sendToken(IERC20(_cfmm), to, lpTokens, s.LP_TOKEN_BALANCE, type(uint256).max) → GammaSwapLibrary.safeTransfer(token, to, amount)`
  - **External/Internal?** External.
  - **Argument control?** `to` and `amount`.
  - **Impact**: The caller can transfer any number of tokens that is less than `s.LP_TOKEN_BALANCE`, but they must return the same or a larger amount after

the `externalCall` function call; it will be checked inside the `payLoanAndRefundLiquidator` function.

- `externalSwap(_loan, s.cfmm, amounts, lpTokens, to, data) → IExternalCallee(to).externalCall(msg.sender, amounts, lpTokens, data);`
  - **External/Internal?** External.
  - **Argument control?** `msg.sender`, `amounts`, `lpTokens`, and `data`.
  - **Impact**: The reentrancy is not possible because the other important external functions have `lock`. If caller does not return enough amount of tokens, the transaction will be reverted.
- `externalSwap(_loan, s.cfmm, amounts, lpTokens, to, data) → updateCollateral(_loan) → GammaSwapLibrary.balanceOf(IERC20(tokens[i]), address(this)); → address(_token).staticcall(abi.encodeWithSelector(_token.balanceOf.selector, _address))`
  - **Impact**: Return the current token balance of this contract. This balance will be compared with the last `tokenBalance[i]` value; if the balance was increased, the `_loan.tokensHeld` and `s.TOKEN_BALANCE` will be increased too. But if the balance was decreased, the withdrawn value will be checked that it is no more than `tokensHeld[i]` (available collateral) and the `_loan.tokensHeld` and `s.TOKEN_BALANCE` will be increased.
- `payLoanAndRefundLiquidator(tokenId, tokensHeld, loanLiquidity, 0, true) -> GammaSwapLibrary.safeTransfer(IERC20(s.cfmm), msg.sender, lpRefund);`
  - **External/Internal?** External.
  - **Argument control?** No.
  - **Impact**: The user should not control the `lpRefund` value. Transfer the remaining part of CFMMLPTokens.

4.2 Module: ExternalLongStrategy.sol

Function: `_rebalanceExternally(uint256 tokenId, uint128[] amounts, uint256 lpTokens, address to, byte[] data)`

Allows the loan's creator to use a flash loan and also rebalance a loan's collateral.

Inputs

- `tokenId`
  - **Validation**: There is a check inside the `_getLoan` function that `msg.sender` is creator of loan.
  - **Impact**: A `tokenId` refers to an existing `_loan`, which will be rebalancing.
- `amounts`

  - **Validation**: There is a check that `amount <= s.TOKEN_BALANCE` inside `externalSwap→sendAndCalcCollateralLPTokens→sendToken` function.
  - **Impact**: Amount of tokens from the pool to flash loan.
- `lpTokens`
  - **Validation**: There is a check that `lpTokens <= s.LP_TOKEN_BALANCE` inside `externalSwap→sendCFMMLPTokens→sendToken` function.
  - **Impact**: Amount of CFMM LP tokens being flash loaned.
- `to`
  - **Validation**: Cannot be zero address.
  - **Impact**: Address that will receive the collateral tokens and/or `lpTokens` in flash loan.
- `data`
  - **Validation**: No checks.
  - **Impact**: Custom user data. It is passed to the `externalCall`.

Branches and code coverage (including function calls)

Intended branches

- [x] `lpTokens` $\neq$ `0`.
- [ ] `amounts` is not empty.
- [x] `amounts` is not empty and `lpTokens` $\neq$ `0`.
- [x] Withdraw one of the tokens by no more than the available number of tokens.
- [x] Withdraw both tokens by no more than the available number of tokens.
- [x] Deposit one of the tokens.
- [x] Deposit both tokens.
- [x] Deposit one token and withdraw another.

Negative behavior

- [x] `_loan` for `tokenId` does not exist.
- [ ] `msg.sender` is not creator of the `_loan`.
- [ ] Balance of contract is not enough to transfer `amounts`.
- [ ] Balance of contract is not enough to transfer `lpTokens`.
- [ ] Zero `to` address.
- [ ] After `externalCall`, the `s.cfmm` balance of the contract has not returned to the previous value.
- [ ] After `externalCall`, the balance of the contract for each `tokens` has not returned to the previous value.
- [ ] After `externalCall`, the balance of the contract for one of `tokens` has not returned to the previous value.

- ☑ Withdraw one of the tokens, and loan is undercollateralized after `externalCall`.
- ☑ Withdraw both tokens, and loan is undercollateralized after `externalCall`.
- ☑ Withdraw one of the tokens and deposit another, and loan is undercollateralized after `externalCall`.
- ☐ The `amounts` and `tokenId` are zero.

Function call analysis

- `externalSwap(_loan, s.cfmm, amounts, lpTokens, to, data) → sendAndCalcCollateralLPTokens(to, amounts, lastCFMMTotalSupply) → sendToken(IERC20(tokens[i]), to, amounts[i], s.TOKEN_BALANCE[i], type(uint128).max) → GammaSwapLibrary.safeTransfer(token, to, amount)`
  - **External/Internal?** External.
  - **Argument control?** `to` and `amount`.
  - **Impact**: The caller can transfer any number of tokens that is less than `s.TOKEN_BALANCE[i]`, but they must return the same or a larger amount after the `externalCall` function call; it will be checked inside the `updateCollateral` function.
- `externalSwap(_loan, s.cfmm, amounts, lpTokens, to, data) → sendCFMMLPTokens(_cfmm, to, lpTokens) → sendToken(IERC20(_cfmm), to, lpTokens, s.LP_TOKEN_BALANCE, type(uint256).max) → GammaSwapLibrary.safeTransfer(token, to, amount)`
  - **External/Internal?** External.
  - **Argument control?** `to` and `amount`.
  - **Impact**: The caller can transfer any number of tokens that is less than `s.LP_TOKEN_BALANCE`, but they must return the same or a larger amount after the `externalCall` function call; it will be checked inside the `checkLPTokens` function.
- `externalSwap(_loan, s.cfmm, amounts, lpTokens, to, data) → IExternalCallee(to).externalCall(msg.sender, amounts, lpTokens, data);`
  - **External/Internal?** External.
  - **Argument control?** `msg.sender`, `amounts`, `lpTokens`, and `data`.
  - **Impact**: The reentrancy is not possible because the other important external functions have `lock`. If caller does not return enough amount of tokens, the transaction will be reverted.
- `externalSwap(_loan, s.cfmm, amounts, lpTokens, to, data) → updateCollateral(_loan) → GammaSwapLibrary.balanceOf(IERC20(tokens[i]), address(this)); → address(_token).staticcall(abi.encodeWithSelector(_token.balanceOf.selector, _address))`
  - **External/Internal?** External.

  - **Argument control?** No.
  - **Impact**: Return the current token balance of this contract. This balance will be compared with the last `tokenBalance[i]` value; if the balance was increased, the `_loan.tokensHeld` and `s.TOKEN_BALANCE` will be increased too. But if the balance was decreased, the withdrawn value will be checked that it is no more than `tokensHeld[i]` (available collateral) and the `_loan.tokensHeld` and `s.TOKEN_BALANCE` will be increased.

- `externalSwap(_loan, s.cfmm, amounts, lpTokens, to, data)` → `checkLPTokens(_cfmm, prevLpTokenBalance, lastCFMMInvariant, lastCFMMTotalSupply)` → `GammaSwapLibrary.balanceOf(IERC20(_cfmm), address(this))`
  - **External/Internal?** External.
  - **Argument control?** No.
  - **Impact**: Return the current `_cfmm` balance of this contract. This new balance will be compared with the balance before the `externalCall` function call, and if new value is less, the transaction will be reverted. Also, update the `s.LP_TOKEN_BALANCE` and `s.LP_INVARIANT`.

5 Audit Results

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

During our audit, we discovered one finding that was informational in nature. GammaSwap 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 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 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.

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.