PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Zealous Zombie Finance

16 October 2022

paladinsec.co

info@paladinsec.co

Table of Contents

Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocation for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

Paladin retains full rights over all intellectual property (including expertise and new attack or exploit vectors) discovered during the audit process. Paladin is therefore allowed and expected to re-use this knowledge in subsequent audits and to inform existing projects that may have similar vulnerabilities. Paladin may, at its discretion, claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for Zealous Zombie Finance on the Optimism network. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

The part of Zealous Zombie Finance that is being audited is a Decentralized Exchange (DEX) project. Users will be able to use the platform as a means of trading their cryptocurrencies in a non-custodial manner without an intermediary.

General risks on DEX projects include:

- Excessive governance privilege where a single address or group of addresses can control the flow of funds in the project contracts (rug risk)
- Contract bugs that can lead to the exploit of miscalculated swap fees or balance calculation
- Not enough liquidity supplied, causing cost and pricing instability
- Front-running risks by trading bots

Below, the audit will cover issues found in the Zealous Zombie Finance protocol and provide recommendations for mitigating them. Users should take note that the audit does not include the periphery contracts (router) of the DEX and we highly advise users to take necessary precautions when interacting with the periphery contracts.

1.1 Summary

Project Name	Zealous Zombie Finance
URL	https://zz.finance/
Network	Optimism
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
ZombieERC20	Dependency of ZombiePair	MATCH
ZombieFactory	0x5Ee7b02aD05BC8d1D4C7e112CE360231225C1a15	MATCH
ZombiePair	0xf719033D0D078b71Bdc2bB43547Ea836d626A715	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	0	-	-	-
Medium	0	-	-	-
Low	0	-	-	-
Informational	11	-	-	11
Total	**11**	**-**	**-**	**11**

Classification of Issues

Severity	Description
High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
Medium	Bugs or issues that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
Informational	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

1.3.1 ZombieERC20

ID	Severity	Summary	Status
01	INFO	Approval event is not emitted if allowance is changed in transferFrom as suggested in the ERC-20 Token Standard (this issue is also present in Uniswap)	ACKNOWLEDGED
02	INFO	`permit` can be frontrun to prevent someone from calling `removeLiquidityWithPermit` (also present in Uniswap)	ACKNOWLEDGED

1.3.2 ZombieFactory

ID	Severity	Summary	Status
03	INFO	Typographical error	ACKNOWLEDGED
04	INFO	`feeToSetter` lacks a non-zero requirement in the constructor	ACKNOWLEDGED
05	INFO	The `uint` variable is used within the whole contract	ACKNOWLEDGED
06	INFO	Lack of events for the governance functions: `setFeeTo` and `setFeeToSetter`	ACKNOWLEDGED

1.3.3 ZombiePair

ID	Severity	Summary	Status
07	INFO	factory can be made immutable	ACKNOWLEDGED
08	INFO	The uint variable is used within the whole contract	ACKNOWLEDGED
09	INFO	token0 and token1 can be cast to IERC20	ACKNOWLEDGED
10	INFO	factory can be cast to IZombieFactory	ACKNOWLEDGED
11	INFO	Pairs without supply but with a partial reserve might crash the frontend if the user wants to swap on this pair (this issue is present in most frontends)	ACKNOWLEDGED

1.3.4 Interfaces & Libraries

No issues found.

2 Findings

2.1 ZombieERC20

`ZombieERC20` is an implementation of the ERC-20 token standard where it represents a share value of a user's assets that is supplied in the liquidity pools. This contract is inherited by `ZombiePair.sol` and it is forked from UniswapV2's `UniswapV2ERC20` core contract.

2.1.1 Issues & Recommendations

Issue #01	**Approval event is not emitted if allowance is changed in `transferFrom` as suggested in the ERC-20 Token Standard (this issue is also present in Uniswap)**
Severity	INFORMATIONAL
Description	The ERC-20 standard specifies that an approval event should be emitted when the allowance of a user changes. However, within the ERC20 implementation of both Uniswap and Zealous Zombie, this is not done. You can read more about this improvement in Pull Request #65 of uniswap-core.
Recommendation	Consider adding `emit Approval(from, msg.sender, remaining)` in `transferFrom` when allowance is modified.
Resolution	ACKNOWLEDGED

Issue #02	**`permit` can be frontrun to prevent someone from calling `removeLiquidityWithPermit` (also present in Uniswap)**
Severity	INFORMATIONAL
Description	Currently if `permit` is executed twice, the second execution will be reverted. It is possible, in theory, for a bot to pick up `permit` transactions in the mempool and execute them before a contract can. The implications of this issue is that a bad actor could prevent a user from removing liquidity with a `permit` through the router. It is a denial of service attack which is present in all AMMs but which we have yet to witness being used since there is no profit from it.
Recommendation	Consider this issue if there are ever complaints by users that their `removeLiquidityWithPermit` transactions are failing. It could be the case that someone is using this vector against them. We do not recommend changing this behavior since it would cause a lot of extra work modifying the frontend to account for the new `permit` behavior. This issue is also present in Uniswap after all.
Resolution	ACKNOWLEDGED

2.2 ZombieFactory

`ZombieFactory` is responsible for keeping track of all existing liquidity pairs of the DEX and allows users to create new ones. Additionally, the contract also stores the necessary addresses where the mint fee is sent to as well as the governance address. This contract is a fork of UniswapV2's `UniswapV2Factory` core contract.

2.2.1 Privileged Functions

- `setFeeTo`
- `setFeeToSetter`

2.2.2 Issues & Recommendations

Issue #03	Typographical error
Severity	INFORMATIONAL
Description	Line 16 `event PairCreated(address indexed token0, address indexed token1, address pair, uint);` The last parameter can be `length`.
Recommendation	Consider fixing the typographical error.
Resolution	ACKNOWLEDGED

Issue #04	`feeToSetter` lacks a non-zero requirement in the constructor
Severity	INFORMATIONAL
Description	During the deployment of the contract, the deployer is able to put in any address as the `feeToSetter` including a zero address. Having a zero address as the `feeToSetter` would practically disable the two governance functions: `setFeeTo` and `setFeeToSetter`.
Recommendation	To prevent this from happening by accident, consider adding a non-zero address requirement to the relevant function.
Resolution	ACKNOWLEDGED

Issue #05	The `uint` variable is used within the whole contract
Severity	INFORMATIONAL
Description	We recommend being consistent and only use `uint256`. Being consistent demonstrates to third-party validators that the code has been carefully thought through.
Recommendation	Consider using `uint256` throughout the contract.
Resolution	ACKNOWLEDGED

Issue #06	Lack of events for the governance functions: `setFeeTo` and `setFeeToSetter`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the above functions.
Resolution	ACKNOWLEDGED

2.3 ZombiePair

`ZombiePair` is responsible for keeping track of pool token balances by issuing ERC20 receipt tokens to users when they supply liquidity to the protocol or burning the tokens when withdrawing liquidity. The contract also works in tandem with the periphery contracts of the DEX when supplying or withdrawing liquidity in the protocol, or when trading assets.

The contract is a fork of UniswapV2's `UniswapV2Pair` core contract but with a slight modification of the LP minting fee and adjusted balance calculation. The LP minting was fee changed from ⅙ of the liquidity to ⅓, and adjusted balance calculation has been changed from 0.3% to 0.25%.

Users should take note that these changes need to have a corresponding update in the periphery contracts as well or these could open up several attack vectors and be used to drain liquidity from the protocol.

2.3.1 Privileged Functions

- `initialize`

2.3.2 Issues & Recommendations

Issue #07	`factory` can be made `immutable`
Severity	INFORMATIONAL
Description	Variables that are only set in the constructor but never modified can be indicated as such with the immutable keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and saves gas.
Recommendation	Consider making `factory` immutable.
Resolution	ACKNOWLEDGED

Issue #08	The uint variable is used within the whole contract
Severity	INFORMATIONAL
Description	We recommend being consistent and only use `uint256`. Consistency demonstrates to third-party validators that the code has been carefully thought through.
Recommendation	Consider using `uint256` throughout the contract.
Resolution	ACKNOWLEDGED

Issue #09	`token0` and `token1` can be cast to `IERC20`
Severity	INFORMATIONAL
Description	Variables that are of type `IERC20` can be marked as this to prevent casting repeatedly in the contract.
Recommendation	Consider marking the above variables of type `IERC20`.
Resolution	ACKNOWLEDGED

Issue #10	`factory` can be cast to `IZombieFactory`
Severity	INFORMATIONAL
Description	Variables that are of type `IZombieFactory` can be marked as this to prevent casting repeatedly in the contract.
Recommendation	Consider marking the above variable of type `IZombieFactory`.
Resolution	ACKNOWLEDGED

Issue #11	**Pairs without supply but with a partial reserve might crash the frontend if the user wants to swap on this pair (this issue is present in most frontends)**
Severity	INFORMATIONAL
Description	A malicious DoS attack we have witnessed in practice is when a project wants to go live through a presale, people can instantiate the pair while there are no tokens yet. The malicious party will then sent some of the counterparty token to this pair so it has a partial balance (eg. 0.1 BNB and 0 tokens). When `sync()` is then called, the pair's reserves are updated to account for this balance. Due to a division by zero exception, many frontends can not properly account for this state and will go through a blank page, preventing the original project from adding liquidity through the frontend.
Recommendation	Consider checking whether this is present in the frontend and adding a division by zero handler.
Resolution	ACKNOWLEDGED

2.4 Interfaces & Libraries

The interfaces and libraries included in the project is identical to the UniswapV2 implementation. There are no significant differences between the two aside from the rebranding and the replacement of the `uint` variable with `uint256`.

2.4.1 Issues & Recommendations

No issues found.


PALADIN
BLOCKCHAIN SECURITY