SMART CONTRACTS

TABLE OF CONTENTS

ABOUT TRUTH SECURITY

Truth Security Audits conducts a comprehensive security review of blockchain applications, using modern tools and employing only the most experienced solidity experts on the market. During the elaboration of the audit, auditors analyze possible attack vectors both from the project owners and its users and rate them by Severity (see Appendix A) from Informational to Critical as per common reason, giving an adequate explanation in the body of the audit. Our Audits are versioned, and clients get a grace period to alleviate or comment on all of our findings.

METHODOLOGY

As per standard code review practices, we use manual and static analysis. During the manual phase, auditor(s) review source code line-by-line, studying its intended and actual behavior, referencing known vulnerabilities (including SWC Registry https://swcregistry.io/), comparing the code to common contracts, and noting all things that are out of the ordinary for confirmation or rejection. Static analysis refers to a computer-aided analysis of the code, providing automatized and powerful insight into additional subtle issues possibly present in the code.

Some auditors additionally write their own test cases and try to break the contracts in their own local simulated blockchain environment. This is called dynamic analysis and is often employed in the more complex contracts, where consolidated testing scenarios help assess the completeness of contract logic or reversely give a proof-of-concept for potential security vulnerability.

Finally, all auditors do on-chain verification of live contracts. This crucial step confirms contracts were not swapped for malicious only after audit, or that parameters are set based on reasonable expectations. Issues such as non-renounced ownerships are also assessed in this step.

PROJECT DETAILS

Project Name	OPTFund
Description	The OPTFund contract is a staking and gambling contract which allows users to stake their OPTV3 tokens and receive yield on these. This contract is solely intended to be used by OptimusV3 and will not work with any sort of transfer-tax tokens. Hence we do not recommend forking this contract and using it for other purposes.
Links	optfund.net
Code Language	Solidity
Chain	Polygon

VERSION DETAILS

Version	Based on status at	Published at	Elaborated by	Notes
V	November 27th, 2022	November 27th, 2022	November 27th, 2022	

OBJECTS OF REVIEW

In Versions	Source	Contents
Version 1		

SUMMARY

AUDIT PROCEDURE

Auditors	RMIII
Audited as	Truth Seekers
Methodology	Static Analysis + Dynamic Analysis Static analysis examines code to identify issues within the logic and techniques. This type of analysis addresses weaknesses in source code that might lead to vulnerabilities. Static analysis supports development operations by creating an automated feedback loop. Static review can lead to false positives/negatives which Truth Seekers confirms or denies with penetration testing. Static analysis is being used on Opt3Fund because it specifically identifies defects before you run a program (e.g, between coding and unit testing)
Tools	Dynamic analysis finds vulnerabilities in a runtime environment. Automated tools analyze the input and output of an application for potential threats like SQL injection. Tools can also search for other application-specific issues and analyze server configuration errors. The purpose of dynamic analysis is to analyze the program as an attacker would, looking for entry points and vulnerable sections during program execution. Hence, we create a well written suite of manually ran unit and regression tests.

Truth Security Audits has no control over website UI projects provide. Always double check you are signing a contract matching one of the contracts in section Objects of Review.

Truth Security Audits concerns itself exclusively with code quality and smart contract security.
We have not audited tokenomics, nor a general likelihood of making money with this project. Truth Security report is not financial advice.

FINDINGS

Finding ID	001	Severity	High
Type	The auditInvestorAddStake function does not set any deposit related	Status	Unresolved
Location	Entirety		
Description	Currently, the aforementioned function lacks setting any deposit related variables like depositor, lastClaim, gambled, etc. This will lead to critical issues especially for lastClaim, consider following scenario: Bob deposits 100 tokens on timestamp 1669053776 Bob doesn't interact with the contract for a week The admin now calls auditInvestorAddStake with 15.000 tokens and Bob’s address Bob calls now claimRewards which will claim the amount for 15.100 tokens based on timestamp 1669053776 Bob now receives way more tokens than he deserves, which will drain the contract. This issue is amplified if Bob deposited even months before It should be noted that Bob must already have deposited, otherwise personalStakingPercentage will be zero which results in zero during the calculateRewards. Additionally, it should be ensured that the privileged address is not in the locked state, otherwise this could lead to potentially bad edge-cases.		
Recommendation	We recommend to follow the same logic as a deposit while executing this function, especially setting lastClaim.		
Alleviation			

Finding ID	002	Severity	High
Type	Governance privilege: auditInvestorStake allows for an arbitrary amount of tokens as parameter	Status	Unresolved
Location	auditInvestorStake		
Description	Currently, the function auditInvestorStake has the following requirement: require(totalAmountAddedAudit <= 15000, "Added amount exceeds what investor paid."); However, the admin can simply input a huge, arbitrary _amount as a parameter which will circumvent this requirement. This can lead to a total loss of user funds, if abused by the admin.		
Recommendation	We recommend changing the requirement as follows: require(totalAmountAddedAudit + _amount <= 15000, "Added amount exceeds what investor paid.");		
Alleviation	-		

Finding ID	003	**Severity**	High
Type	devWallet can be set to address(0)	**Status**	Unresolved

Location	devWallet
Description	Currently, the function changeDevWallet lacks a non-zero check. If the devWallet is ever set to address(0), this will block all withdrawals, resulting in stuck user funds.
Recommendation	We recommend adding a non-zero validation for the aforementioned function.
Alleviation	

Finding ID	004	**Severity**	High
Type	The fee calculation within depositOPT3 is flawed	**Status**	Unresolved

Location	depositOpt3
Description	The depositopt3 function calculates the fees based on amount: uint256 depositFee = (_amount * stakingFee).div(percentRate); uint256 depositAfterFees = _amount.sub(depositFee); uint256 amountForDev = (_amount * devFee).div(percentRate); uint256 amountForBurn = (_amount * OPT3BurnFee).div(percentRate); **This will become a problem if devFee and OPT3BurnFee combined get higher than stakingFee** This mechanism can also be abused by the admin to drain the contract.
Recommendation	Our recommendation is to calculate devFee and burnFee based on depositFee.
Alleviation	

Finding ID	005	**Severity**	Medium
Type	Investors might get locked in gambling mode without receiving a change of personalStakingPercentage	**Status**	Unresolved
Location	personalStakingPercentage		
Description	Currently, the ChainLink contract calls the function rawFulfillRandomwords which then internally calls fulfillRandomWords. This function is responsible to set personalStakingPercentage, however, if the subscription is not sufficiently funded, this function will never be called, leaving the investor in a locked state until lockTime is passed, without any change in personalStakingPercentage		
Recommendation	We recommend to always ensure that the subscription is funded, moreover it might make sense to implement a governance function to reset the locked status of an investor. However, this is not necessary and just an additional safeguard.		
Alleviation			

Finding ID	006	**Severity**	Medium
Type	AuditInvestorStake does not transfer any tokens in	**Status**	Unresolved
Location			
Description	The aforementioned function simply assigns the _amount to the provided address, however, it does not transfer any tokens to the contract. This means that all investors will have to bear the loss, which is especially bad in the following scenario: Bob deposits 15.000 tokens Admin calls auditInvestorStake with Alice’s address and 15.000 tokens Alice withdraws 15.000 tokens Bob is left empty and loses all his funds.		
Recommendation	We recommend transferring tokens to the contract while executing this function.		
Alleviation			

Finding ID	007	**Severity**	Low
Type	Global: Token is not defined with decimals	**Status**	Unresolved
Location			
Description	Currently, the contract does not define token amounts with decimals, which means that all requirements like require(totalAmountAddedAudit <= 15000, "Added amount exceeds what investor paid."); will result in undesired side-effects.		
Recommendation	We recommend adding the necessary decimals (1e18).		
Alleviation			

Finding ID	008	**Severity**	Low
Type	fulfillRandomWords lacks fulfilled check	**Status**	Unresolved
Location			
Description	Currently, the fulfillRandomWords function does not check if the requestId is already fulfilled, it only sets fulfilled = true afterwards. At the current state it doesn't seem that fulfillRandomWords can be called twice for the same requestId, however, to align with the contract logic, a validation should occur before the function gets executed.		
Recommendation	We recommend a simple requestMapping[requestId].fulfilled == false; requirement at the beginning of the function.		
Alleviation			

Finding ID	009	**Severity**	Informational
Type	Contract does not adhere to checks-effects-interactions/ ResetGamble lacks explicit error messages	**Status**	Unresolved

Location	
Description	Writing secure smart-contracts requires to adhere to several best-practices. One of these is the checks-effects-interactions pattern (https://fravoll.github.io/solidity-patterns/checks_effects_interactions.html) For example the depositOPT3 function does not adhere to this pattern. ResetGamble: During our tests, we realized that the function resetGamble does not provide an explicit error statement if the deposit unlockTime is not exceeded. require(block.timestamp >= depo.lockedUntil); This will potentially confuse third-party auditors as well as users.
Recommendation	We recommend to adhere to this pattern whenever possible to avoid any potential reentrancy attacks. ResetGamble: Our recommendation is to add explicit error messages.
Alleviation	

Finding ID	010	**Severity**	Informational
Type	Functions/Variables	**Status**	Unresolved

Location	
Description	Functions that are only called externally and not used within the contract can be made external. This can save some gas. Functions that manipulate the contract state should emit events. Variables that are important for the contract's business logic should be easily inspectable by users. Variables that are defined in the constructor and never changed can be made immutable to save gas. Variables that are defined and never changed can be marked as CONSTANT, this will save some gas. Variables that serve a special purpose should also be declared as this purpose, this keeps the contract clean and readable. Variables, functions, imports etc. which are not used within the contract can be removed. This reduces the contract size and keeps the code clean.
Recommendation	We recommend marking the following variables explicitly private: keyHash, s_subscriptionId, COORDINATOR, callbackGasLimit, requestConfirmations, numWords, devWallet, totalAmountAddedAudit. marking the following variables as immutable: keyHash, s_subscriptionId, COORDINATOR, We recommend marking the following variables as constant: callbackGasLimit, requestConfirmations, numWords, rewardPeriod, percentRate. Change Variable uint256 lastRoiTime = block.timestamp - deposit[_address].lastClaim; to secondsPassed instead of lastRoiTime.
Alleviation	

APPENDIX A: ISSUE SEVERITY CLASSIFICATION

The Auditor(s) assign one of the following Severities to every finding as per common practice.

Critical. Such issues may result in significant loss of funds, complete contract logic breakdown, or the ability of project owners to withdraw liquidity in an unreasonable way. Code snippets proving the project owner's malicious intent are flagged as critical as well. Critical issues require immediate attention, and investing in projects with critical issues is extremely risky. Examples include unguarded mint functions or their executions, provably illicit pool drainage logic, or potential flash-loan vulnerabilities.

Major. Although not proving malicious intent by themselves, major issues may still be exploited by project owners or users for a significant loss of funds or very irregular contract behavior. Examples include centralized ownership without Timelocks and multi-signatures, potential reentrancy vulnerabilities, and concentrated holdings of tokens.

Medium. Such issues do not pose an immediate and severe risk but may pose a risk of partial loss of funds or irregular contract behavior. Examples include susceptibility to obviously unintended investment strategies, high-impact integer overflows, or high-impact standardization faults such as library usage,

Minor. These issues pose a low risk to contract logic or investor funds but may be convenient to consider. Examples include integer overflows in non-essential places, nonversioned libraries, missing or faulty licensing, misleading function names, or low-impact standardization mistakes.

Informational. These issues do not pose any risk to contract logic and investor funds, Examples include tokenomics clarifications, gas optimizations, redundant code, misleading comments, style, and convention.

Confirmational. In specific situations, we issue these findings, which confirm some of the universally-concerning facts that many investors seek. Examples include contract renounces and confirmation of a contract being fork of another protocol. Note: These points are not actual issues. Obviously, only a small subset of tests ran in an audit suite receives its Confirmational Finding.

APPENDIX B: DISCLAIMER

This audit is for informational purposes only and does not provide any financial or investment advice. This report does not substitute, in any way, due diligence and your own research. This report represents result extensive process intending to help our customers improve quality of their code and readers to assess quality of customers' code, but should not be used in any way to make decisions around involvement in any particular project.

Audit has been done in accordance to methodology as outlined in AboutTruth Security and Audit Procedure sections Unless explicitly and specifically stated, only code quality has been reviewed, focusing on security flaws which could cause loss of funds or logical breakdowns within the contracts. Unless explicitly stated, tokenomics have not been reviewed (although in cases of forks of one project, Auditor may point out cases of significant deviations from common settings). Website UI has not been reviewed, as it is impossible for any auditing body to assure security of domains which are under absolute control of owners - always check you are signing correct contracts.

The report does not signify an approval," „endorsement," or ,,disapproval of the Project The audit does not indicate in any way your likelihood of making, or not losing, money in the project, as we have no control over issues such as general viability of financial primitives presented, their tokenomics, and actions of project owners including, but not limited to, selling their positions or abandoning the project.

The audit has been based on status dated in section Version Details, on artifacts detailed in Objects of Review. Specifically, we have no control nor knowledge of changes made after the date, or on different artifacts. In case the Objects of Review are not live contracts, but private code or GitHub repositories, we expect these artifacts to be full, unaltered, unabridged, and not misleading.

The audit has been elaborated by paid professional(s) as mentioned in section Audit Procedure. Please note that all statements made in this report are Auditor(s)' and do not reflect stance of © Truth Security Audits itself.