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	OptimusV3
Description	Optimus is a decentralized algorithmic trading protocol secured by the Polygon blockchain. Our protocol allows users to benefit from algorithmic trading without the hassle of making your own bots or using a sketchy third-party API linked to your identity on a centralized exchange. Optimus is a low-risk algorithmic trading solution that generates profits for OPT3 holders on a daily basis. The profits of Optimus' trades - averaging 1% per day - are distributed to OPT3 holders automatically. This simplifies the user experience and makes the power of AI trading accessible to everybody. Taking part is easy: just buy OPT3 and hold it in your wallet to earn 1% daily.
Links	https://optimus.money/
Code Language	Solidity
Chain	Polygon

VERSION DETAILS

Version	Based on status at	Published at	Elaborated by	Notes
V	November 8th, 2022	November 8th, 2022	November 8th, 2022	

OBJECTS OF REVIEW

In Versions	Source	Contents
V		
		Optimus V3 Token: 0xCf630283E8Ff2e30C29093bC8aa58CADD8613039

SUMMARY

AUDIT PROCEDURE

Auditors	RMIII
Audited as	Truth Seekers
Methodology	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 Optimus 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	Informational
Type	Contract will completely break for one epoch	Status	Resolved
Location	Line 487		
Description	The coreRebase function has a minor issue in line 487: It only checks if the old_supply is larger than the MAX_SUPPLY which results in new_supply exceeding MAX_SUPPLY at some point in the future. Once that happens the divisor which is set in line 494 will become zero. When the divisor then is zero, the contract will break in various places like _transfer, where the _beforeTokenTransfer reverts due a division by zero; _approve, where the event emitting reverts due to a division by zero etc. At this point the contract will break completely, until either the admin calls adminRebase, or one day is passed and then manualRebase is called; these functions will set the divisor to one making all functionality possible again. **Under normal operation (where the contract rebases at a rate of 1% daily) this error will not occur until 24 years from now.**		
Recommendation	The solution is to rewrite the logic in line 487 to check if old_supply + supplyDelta exceeds MAX_SUPPLY, and in that case simply set new_supply to MAX_SUPPLY.		
Alleviation	Acknowledged. The developer will under no circumstances increase the supply dramatically by amplifying adminRebase, which prevents that potential state. Will be fixed in V4.		

Finding ID	002	Severity	Informational
Type	AdminRebase	Status	Resolved
Location	Whole		
Description	Even if the admin has no possibility to mint new tokens and the AdminRebase is for syncing new versions to previous versions of OPT. However, AdminRebase presents a potential vulnerability because it could be used in tandem with setRewardYield in a way that would allow a lot of tokens to be created.		
Recommendation	Patch in V4.		
Alleviation	The vulnerability will be patched in v4 by imposing reasonable limits on the setRewardYield function. -		

Finding ID	003	**Severity**	Informational
Type	AdminRebase	**Status**	Resolved
Location	Whole		
Description	adminRebase is not required after initialization of the contract. In V4, we will eliminate this attack vector by disabling adminRebase after contract initialization.		
Recommendation	adminRebase will not be called after initialization, so there is no external attack vector		
Alleviation	Resolved.		

Finding ID	004	**Severity**	Low
Type	divisor rounds down	**Status**	Resolved
Location	Line 494:		
Description	This happens in line 494: _setDivisor(MAX_SUPPLY.div(new_supply)); One can simply input any arbitrary amounts for MAX_SUPPLY and new_supply which are near each other: _setDivisor((1000000*1e30) / (300000*1e30)) > result is 3.33333 but solidity rounds it down to 1 If new_supply increases now, the result in the next round will also be 3 due to solidity rounding down.. **Under normal operation (where the contract rebases at a rate of 1% daily) this error will not occur until 24 years from now."**		
Recommendation	The solution is to implement a logic that keeps the divisor always reasonably large to prevent rounding issues.		
Alleviation	Acknowledged. The developer will under no circumstances increase the supply dramatically by amplifying adminRebase, which avoids that potential state.		

Finding ID	005	**Severity**	Medium
Type	The admin can block all second transfers within one day	**Status**	Unresolved
Location	Whole		
Description	The admin has the ability to change limit within the function setLimit. If that limit is set to zero, the function iterate will always trigger setCooldown, which effectively blocks the address from further transfers for the next 5 days (or forever if the admin changes toLength via setLength.		
Recommendation	The solution is to add reasonable safeguards for the setter functions		
Alleviation	The next version will contain the fixes.		

Finding ID	006	**Severity**	Informational
Type	limiter modifier can be circumvented	**Status**	Unresolved
Location	Whole		
Description	Currently, the contract has an upper limit for transfers during one epoch (one day). This limit can get simply circumvented by either transferring the funds between accounts, or create a contract with various privileged addresses to call it for sells, buys, transfers etc, since tx.origin will then be the privileged address instead of the contract. **This is widely known aspect that affects majority of contracts.**		
Recommendation	There is no easy fix for that issue since during sells and buys, the router executes the _transfer, which means it is a bad idea to also limit the msg.sender.		
Alleviation	Since there is no method to avoid this, it will get acknowledged.		

Finding ID	007	**Severity**	Low
Type	Various variables are private	**Status**	Unresolved

Location	Lines: 177-179

Description	Private variables are not easily fetchable for average users which effectively decreases the project transparency, especially when these variables are important.
	
	177: uint256 internal limit;
	178: uint256 internal toLength;
	179: bool internal useLimiter;

Recommendation	The solution is to make these variables public.
Alleviation	The next version will contain the fixes

Finding ID	008	**Severity**	Informational
Type	Various variables can be made constant	**Status**	Unresolved

Location	L: 422-423

Description	Variables which are never modified within the contract can be made constant, this includes them directly into the bytecode and safes gas.
	
	422: uint256 public rewardYieldDenominator= 1000;
	423: uint256 public rebaseFrequency = 1 days;

Recommendation	The solution is to make these variables constant
Alleviation	The next version will contain the fixes.

Finding ID	009	**Severity**	Informational
Type	Authorized addresses can not be fetched by users	**Status**	Unresolved
Location	Whole		
Description	The contract does not save any authorized address in an array, which makes it very hard for the average user to fetch these addresses.		
Recommendation	The solution is to push all privileged addresses into an array.		
Alleviation	The developer decided on personal preference to keep it as is.		

Finding ID		**Severity**	
Type		**Status**	
Location			
Description			
Recommendation			
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 Ul 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.