verichains

SECURITY AUDIT OF

KOPS SMART CONTRACTS

Public Report

Oct 09, 2025

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Oct 09, 2025. We would like to thank the KOPS for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the KOPS Smart Contracts. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified some vulnerable issues in the smart contract code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About KOPS Smart Contracts

DeFi offers incredible opportunities, but navigating it can be overwhelming. Finding the best yields, managing risk, and executing trades efficiently requires constant attention—leaving most people feeling one step behind.

This complexity creates a gap between professional traders and everyday users, undermining the promise of open, accessible finance.

KOPS is built to close that gap.

KOPS provides you with a personal, AI-powered agent that puts your crypto to work for you, 24/7. Think of it as hiring a brilliant financial analyst that never sleeps. Your agent intelligently navigates the complexities of DeFi—finding the best yields, managing lending positions, and running advanced strategies—all while you maintain complete control of your funds.

1.2. Audit Scope

This audit focused on identifying security flaws in code and the design of the KOPS Smart Contracts. It was conducted on commit `b7d4fdfe370a7822127bf66c35918bb783b774ff` from git repository: *https://github.com/KOPs-ai/strategy.contracts/blob/main/src/MaxYieldUSDT.sol*

SHA256 Sum	File
`b1f178cfe4e3eab3f6ebae05bca2987e3105bb1b9c6f9160bb2efb9f390f77b4`	`./MaxYieldUSDT.sol`

1.3. Audit Methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that were considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions

- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

KOPS acknowledges that the security services provided by Verichains, are conducted to the best of their professional abilities but cannot guarantee 100% coverage of all security vulnerabilities. KOPS understands and accepts that despite rigorous auditing, certain vulnerabilities may remain undetected. Therefore, KOPS agrees that Verichains shall not be held responsible or liable, and shall not be charged for any hacking incidents that occur due to security vulnerabilities not identified during the audit process.

1.5. Acceptance Minute

This final report served by Verichains to the KOPS will be considered an Acceptance Minute. Within 7 days, if no any further responses or reports is received from the KOPS, the final report will be considered fully accepted by the KOPS without the signature.

2. AUDIT RESULT

2.1. Overview

The KOPS Smart Contracts was written in `Solidity` language, with the required version to be `^0.8.27`. The source code was written based on **OpenZeppelin's library**.

The `MaxYieldUSDT` contract provides a unified execution interface through its main `execute()` function, which serves as the single entry point for all protocol interactions. This function accepts an Action enum parameter that determines which of the four supported operations to perform, along with encoded data containing the specific parameters for each operation.

The contract supports two primary operation types: supply operations and withdraw operations, each available for both the **Hypurrfi** and **Hyperlend** pools.

2.2. Findings

#	Title	Severity	Status
1	Unsafe transfer mechanism	MEDIUM	Fixed

2.2.1. Unsafe transfer mechanism MEDIUM

Position:

- `src/MaxYieldUSDT.sol`

Description:

All the internal supply and withdraw functions call `transferFrom`, `transfer`, and `approve` on USDT/aTokens without checking return values. Some `ERC20` tokens like USDT return false on failure instead of reverting. This allows the function to continue execution even though the `transfer`/`approval` failed, leading to inconsistent balances and stuck funds.

RECOMMENDATION

Wrap the relevant tokens with `SafeERC20` and replace raw calls with `safeTransfer`, `safeTransferFrom`, and `safeApprove`.

UPDATES

- ***Oct 9, 2025*:** The team has acknowledged and implemented the change to the transfer mechanism.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Oct 07, 2025*	Public Report	Verichains Lab
1.1	*Oct 09, 2025*	Public Report	Verichains Lab

Table 2. Report versions history