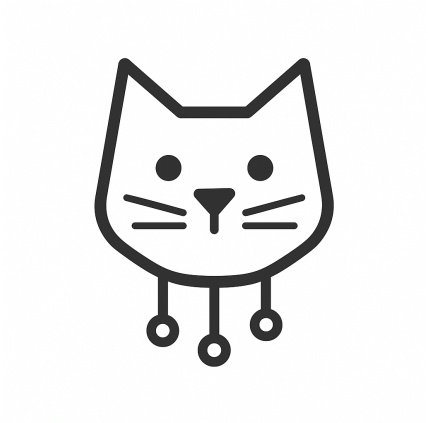
Hermes AI Audit Report

Radciper

June 30, 2025



Hermes AI Audit Report

Version 1.0

Radcipher

June 30, 2025

Hermes AI Audit Report

Radciper

June 30, 2025

Prepared by: [Radcipher]

Table of Contents

Protocol Summary

Hermes AI is a decentralized, community-first investment intelligence protocol powered by a 10-factor AI-driven scoring engine. It delivers real-time, on-chain dividend distribution while analyzing opportunities across NASDAQ, BIST100, and the crypto markets. Backed by GPT-4o for advanced insight generation, Hermes AI combines traditional finance and Web3 into a unified platform designed to help users make smarter, AI-assisted investment decisions.

Disclaimer

Radcipher team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an

endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

Likelihood	Impact		
	High	Medium	Low
High	H	H/M	M
Medium	H/M	M	M/L
Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

- Token: bscscan
- In Scope: ***HermesAIInvestmentFund***

Scope

```
./src/
#-- HermesAIInvestmentFund
```

Executive Summary

The audit of the Hermes AI smart contract identified several best practices and showed a solid implementation with no critical vulnerabilities discovered. The codebase follows standard patterns for decentralized finance applications and demonstrates a clear focus on transparency and security.

Issues found

Severity	Number of issues found
High	0
Medium	0
Low	0
Info	2
Gas	0
Total	2

About

Radcipher is a comprehensive Web3 security firm made up of passionate smart contract researchers, tokenomics experts, and professionals deeply versed in the logic and economic principles behind blockchain protocols. Beyond identifying and addressing critical vulnerabilities in DeFi and NFT projects, we bring a holistic approach combining technical rigor with economic insight. Our team is dedicated to delivering thorough, high-quality security audits that not only focus on bugs but also on the soundness of the underlying protocol logic and tokenomics design. While absolute security can never be guaranteed, we commit to applying our full expertise and meticulous attention to detail to help protect and strengthen your project's foundation. Discover more about our work Radcipher website or connect with us on X @Radcipher.

Disclaimer

No smart contract security review can ensure the complete elimination of vulnerabilities. These reviews are limited by time, resources, and expertise, aiming to identify as many issues as feasible within those constraints. We cannot promise total security following the audit, nor can we guarantee that all problems will be uncovered. We strongly advise implementing ongoing security measures such as additional audits, bug bounty programs, and live on-chain monitoring.

Introduction

Radcipher performed smart contract security assessment of the **HermesAIInvestmentFund** , concentrating on the security aspects of its smart contract implementation.

About Hermes AI

Hermes AI is a next-generation investment protocol that merges artificial intelligence with decentralized technology. Central to its design is a sophisticated 10-factor evaluation engine that analyzes assets across both traditional financial markets and the crypto ecosystem. # Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact Definitions

- **High** – Results in significant material loss to the protocol or causes substantial harm to a large group of users.
- **Medium** – Causes moderate material loss to the protocol or moderately affects a group of users.
- **Low** – Leads to minor material loss or impacts a small number of users.

Recommended Actions Based on Severity

- **Critical** – Immediate fix required, especially if the contract is already deployed.
- **High** – Must be resolved prior to deployment.
- **Medium** – Recommended to fix.
- **Low / Informational** – Optional fix.

Findings

[I-1] Improper Order of Operations in `transferFrom`

Description

In the `transferFrom` function, the contract performs the `_transfer` operation before checking if the caller has sufficient allowance. This violates the Checks-Effects-Interactions pattern, which is critical for ensuring secure and predictable smart contract behavior. By transferring tokens before confirming allowance,

the contract allows for state changes (i.e. balances) to occur before the permission check, which can lead to unnecessary gas consumption and potential reverts after state has already been modified.

```
function transferFrom(address sender, address recipient, uint256 amount)
public override returns (bool) {
    _transfer(sender, recipient, amount);
    _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(
      amount, "transfer amount exceeds allowance"));
    return true;
}
```

Impact

This issue introduces unnecessary risk and inefficiency: If the allowance is insufficient, the transaction reverts after balances have been modified, wasting gas and breaking composability expectations. This pattern can be problematic when the transferFrom call is part of a more complex on-chain interaction, potentially disrupting upstream logic or contracts relying on it. Although not a critical vulnerability, this is a clear deviation from best practices and could lead to unexpected behavior or higher gas costs for users and integrators.

Recommendations

Reorder the logic in the transferFrom function to first check and reduce the allowance, and only then perform the transfer. This ensures the contract adheres to the Checks-Effects-Interactions pattern and avoids unnecessary computation on failing transactions.

[I-2] Presence of Dead Code and Unused Functions After Ownership Renouncement

Description

The contract contains several functions—such as `excludeAccount`, `includeAccount`, and `setAsCharityAccount`—which are restricted to the contract owner. Since the ownership has been renounced by setting the owner address to zero, these functions can no longer be called or used. Consequently, any logic dependent on these functions, including transfer functions handling excluded accounts (`_transferFromExcluded`, `_transferToExcluded`, `_transferBothExcluded`), is effectively dead code. Because no accounts can be excluded or included post-renouncement, these functions will never be executed during the token's lifetime.

```
function excludeAccount(address account) external onlyOwner() {
    require(!_isExcluded[account], "Account is already excluded");
    if(_rOwned[account] > 0) {
        _tOwned[account] = tokenFromReflection(_rOwned[account]);
    }
    _isExcluded[account] = true;
    _excluded.push(account);
}

function includeAccount(address account) external onlyOwner() {
    require(_isExcluded[account], "Account is already included");
    for (uint256 i = 0; i < _excluded.length; i++) {
        if (_excluded[i] == account) {
            _excluded[i] = _excluded[_excluded.length - 1];
            _tOwned[account] = 0;
            _isExcluded[account] = false;
            _excluded.pop();
```

```
            break;
        }
    }
}

function setAsCharityAccount(address account) external onlyOwner() {
      FeeAddress = account;
}

function _transferFromExcluded(address sender, address recipient,
uint256 tAmount) private {
    uint256 currentRate =  _getRate();
    (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount,
     uint256 tFee, uint256 tBurn, uint256 tCharity) = _getValues(tAmount);
    uint256 rBurn =  tBurn.mul(currentRate);
    _excludedToTransferContent(sender, recipient,
     tAmount, rAmount, rTransferAmount);
    _sendToCharity(tCharity, sender);
    _reflectFee(rFee, rBurn, tFee, tBurn, tCharity);
    emit Transfer(sender, recipient, tTransferAmount);
}

function _transferToExcluded(address sender, address recipient,
uint256 tAmount) private {
          uint256 currentRate =  _getRate();
          (uint256 rAmount, uint256 rTransferAmount, uint256 rFee,
          uint256 tTransferAmount, uint256 tFee, uint256 tBurn,
          uint256 tCharity) = _getValues(tAmount);
          uint256 rBurn =  tBurn.mul(currentRate);
          _excludedFromTransferContent(sender, recipient, tTransferAmount,
          rAmount, rTransferAmount);
          _sendToCharity(tCharity, sender);
          _reflectFee(rFee, rBurn, tFee, tBurn, tCharity);
          emit Transfer(sender, recipient, tTransferAmount);
      }

function _transferBothExcluded(address sender, address recipient,
uint256 tAmount) private {
          uint256 currentRate =  _getRate();
          (uint256 rAmount, uint256 rTransferAmount, uint256 rFee,
          uint256 tTransferAmount, uint256 tFee, uint256 tBurn,
          uint256 tCharity) = _getValues(tAmount);
          uint256 rBurn =  tBurn.mul(currentRate);
          _bothTransferContent(sender, recipient, tAmount, rAmount,
          tTransferAmount, rTransferAmount);
          _sendToCharity(tCharity, sender);
          _reflectFee(rFee, rBurn, tFee, tBurn, tCharity);
          emit Transfer(sender, recipient, tTransferAmount);
      }
```

Impact

While this does not introduce a security vulnerability, the presence of dead code leads to: Increased contract size and unnecessary gas costs during deployment. Potential confusion for developers or auditors reviewing

the code, as unused functions may imply incomplete or abandoned features. Reduced maintainability and clarity of the codebase. The unused logic also means that the contract's fee/exclusion mechanism is static and unchangeable, which may limit flexibility for future governance or fee adjustments.

Recommendations

Consider removing or refactoring the unused owner-only functions and related transfer variants to reduce contract complexity and gas costs. If the exclusion functionality is intended to be disabled permanently due to renounced ownership, document this explicitly in the contract comments and project documentation to avoid confusion. If future upgrades or proxy patterns are planned, ensure that any owner-only controls and exclusion mechanisms are reconsidered or redesigned to allow maintainability without introducing dead code.

[Note] Check and verify claimed 75% liquidity lock

The documentation states that 75% of the total token supply (1 trillion HERMES) is locked on PancakeSwap as liquidity. However, we were only able to verify a transaction that transfers 25% of the supply to the liquidity lock: BSCscan transaction

We could not verify the remaining 50% of the supply claimed to be locked. This could be due to a variety of reasons, such as multiple lock transactions, different locking mechanisms, or timing of the lock. Regardless, it's good practice to verify this from our side for transparency and completeness.

Recommendation: Please verify from your side the transaction hash or contract address where the remaining 50% of the supply is locked to support the stated tokenomics.