Cyberscope

Audit Report

ThermAI

May 2025

Repository	https://github.com/ThermAI-Organisation/token-audit
Repository	aa591f730a2997210081212a9fc50f7fedbcfc0b
Audited by	© cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Acknowledged
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	UTPD	Unverified Third Party Dependencies	Acknowledged
●	ISF	Incomplete Setter Function	Acknowledged
●	DTL	Duplicated Transfer Logic	Acknowledged
●	IEE	Inconsistent Event Emission	Acknowledged
●	MU	Modifiers Usage	Acknowledged

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
● Critical	Highly Likely / High Impact
● Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
● Minor / Informative	Unlikely / Low to no Impact

Review

Repository	https://github.com/ThermAI-Organisation/token-audit
Commit	aa591f730a2997210081212a9fc50f7fedbcfc0b
Badge Eligibility	Must Fix Criticals

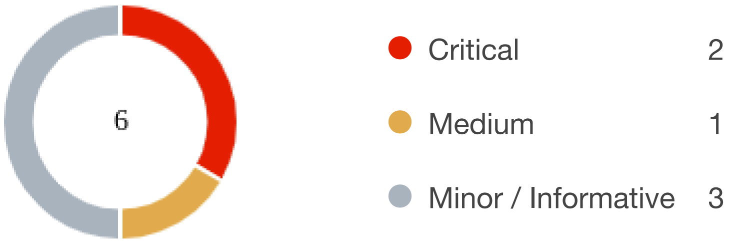
Audit Updates

Initial Audit	24 May 2025
Corrected Phase 2	27 May 2025

Source Files

Filename	SHA256
contracts/interfaces/IThermAITreasury.sol	cf3ebf0b3e9e33f236036744a39351f9eb6df7dc0130b2cddd0f255dd2beefb9
contracts/interfaces/IThermAIToken.sol	c6634c85155b8b89457c4ec874fab41edec6e4833f71566f4ad6f90179739a14
contracts/implementations/TokenImplementation.sol	67d17e455ddc4d603e3d5330ad07fbc5a700440e14c2e272661ba7152b5a60c3

Findings Breakdown



Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	0	2	0	0
● Medium	0	1	0	0
● Minor / Informative	0	3	0	0

ST - Stops Transactions

Criticality	Critical
Location	contracts/implementations/TokenImplementation.sol#L190,214,236
Status	Acknowledged

Description

The `PAUSER_ROLE` has the authority to stop the sales for all users including the owner. The owner may take advantage of it by calling the `pause` function. As a result, the contract may operate as a honeypot.

```
   function transfer(address to, uint256 amount) public
override(ERC20Upgradeable, IERC20) nonReentrant returns (bool) {
        require(!paused(), "Token transfers are paused");
      ...
      }

      function transferFrom(address from, address to, uint256 amount)
public override(ERC20Upgradeable, IERC20) nonReentrant returns (bool) {
        require(!paused(), "Token transfers are paused");
    ...
    }

    function pause() external onlyRole(PAUSER_ROLE) {
        _pause();
    }

    function unpause() external onlyRole(PAUSER_ROLE) {
        _unpause();
    }
```

Recommendation

The contract could embody a check for not allowing passing the contract. The team should carefully manage the private keys of the `PAUSER_ROLE` account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

Team Update

The team has acknowledged that this is not a security issue and states:

We will retain this functionality for emergency safety.

- *PAUSER_ROLE will be assigned to a dedicated multisig wallet with no timelock to allow for rapid response during emergencies.*
- *DEFAULT_ADMIN_ROLE controlled by a multisig behind a timelock, ensuring transparent and deliberate changes to sensitive system parameters. (This will occur post-sale)*
- *This ensures quick mitigation capability without compromising on centralized control.*

UTPD - Unverified Third Party Dependencies

Criticality	Critical
Location	contracts/implementations/TokenImplementation.sol#L226
Status	Acknowledged

Description

The contract uses an external contract in order to determine the transaction's flow. The external contract is untrusted. As a result, it may produce security issues and harm the transactions.

```
try IThermAITreasury(feeRecipient).receiveFees(feeAmount)
```

Recommendation

The contract should use a trusted external source. A trusted source could be either a commonly recognized or an audited contract. The pointing addresses should not be able to change after the initialization.

Team Update

The team has acknowledged that this is not a security issue and states:

- *Added comprehensive documentation in code comments identifying Treasury relationship*
- *Implemented one-time-set mechanism with isFeeRecipientLocked flag*
- *Added lockFeeRecipient() function for post-deployment security*
- *Enhanced setFeeRecipient() with lock check and documentation*
- *Added view function isFeeRecipientLockedStatus() for transparency*

ISF - Incomplete Setter Function

Criticality	Medium
Location	contracts/implementations/TokenImplementation.sol#L100
Status	Acknowledged

Description

The contract is exposing a `setStakingContractAddress` function, which by its naming implies that it will set and store the address of a designated staking contract. However, the function merely excludes the provided address from fee deductions, an action that is already supported by the existing `excludeFromFees` function. There is no storage or tracking of the staking contract address, nor does the function assign this address to any variable within the contract. This indicates that the function is incomplete or misleadingly named, as it lacks its core purpose of setting a staking-related address. Furthermore, it introduces unnecessary redundancy in functionality and may cause confusion for developers or integrators relying on proper staking integration.

```
   function setStakingContractAddress(address stakingContract) external
onlyRole(FEE_MANAGER_ROLE) {
        require(stakingContract != address(0), "Cannot set zero
address");
        _isExcludedFromFees[stakingContract] = true;
        emit ExcludeFromFees(stakingContract, true);
    }

    function excludeFromFees(address account, bool excluded) external
override onlyRole(FEE_MANAGER_ROLE) {
        _isExcludedFromFees[account] = excluded;
        emit ExcludeFromFees(account, excluded);
    }
```

Recommendation

It is recommended to remove the `setStakingContractAddress` function, as its current implementation does not fulfil its intended purpose and duplicates logic already available via `excludeFromFees`. Alternatively, if the staking contract address is intended to be used elsewhere in the contract logic, the function should be updated to store the address in a dedicated state variable for future reference.

DTL - Duplicated Transfer Logic

Criticality	Minor / Informative
Location	contracts/implementations/TokenImplementation.sol#L189,213
Status	Acknowledged

Description

The contract is duplicating identical logic in both the `transfer` and `transferFrom` functions. Each function performs the same core actions, such as pausing checks, fee calculations, conditional fee transfers, and treasury notifications. This results in a redundant codebase where changes or fixes would need to be manually replicated in both functions, increasing the risk of inconsistencies or maintenance errors over time. The duplication also negatively impacts readability and overall contract clarity, especially when future updates are required.

```
 function transfer(address to, uint256 amount) public
override(ERC20Upgradeable, IERC20) nonReentrant returns (bool) {
        ...
    }

    function transferFrom(address from, address to, uint256 amount)
public override(ERC20Upgradeable, IERC20) nonReentrant returns (bool) {
        ...
    }
```

Recommendation

It is recommended to consolidate the shared logic of `transfer` and `transferFrom` into a single internal transfer function. This internal function should handle all common behaviours such as fee calculation, fund transfers, and treasury notification. Both `transfer` and `transferFrom` can then invoke this function, passing only the necessary parameters, improving maintainability and reducing code duplication.

IEE - Inconsistent Event Emission

Criticality	Minor / Informative
Location	contracts/implementations/TokenImplementation.sol#L204,228
Status	Acknowledged

Description

The contract is emitting the `FeesProcessingFailed` event exclusively in the `transfer` function when the fee forwarding call to the treasury fails. However, in the `transferFrom` function, the same fee forwarding logic is executed without emitting the corresponding event in the event of failure. This inconsistency introduces challenges in debugging and monitoring, as failed fee processing events from `transferFrom` transactions remain silent and undetected. Consistent event emission is crucial for off-chain systems, monitoring tools, and transparency, especially when both functions perform nearly identical operations.

```
    function transfer(address to, uint256 amount) public
override(ERC20Upgradeable, IERC20) nonReentrant returns (bool) {
        ...
            // Notify trusted ThermAI Treasury of received fees
            try IThermAITreasury(feeRecipient).receiveFees(feeAmount) {
                // Successfully called receiveFees on trusted Treasury
            } catch (bytes memory reason) {
                // Log the error for debugging
                emit FeesProcessingFailed(feeAmount, reason);
        ...
    }

     function transferFrom(address from, address to, uint256 amount)
public override(ERC20Upgradeable, IERC20) nonReentrant returns (bool) {
        ...
            // Notify trusted ThermAI Treasury of received fees
            try IThermAITreasury(feeRecipient).receiveFees(feeAmount) {}
catch {}
        } else {
            super._trasfer(from, to, amount);
        }
        ...
}
```

Recommendation

It is recommended to emit the `FeesProcessingFailed` event in the `transferFrom` function as well, mirroring the behaviour of the `transfer` function. This ensures consistency across the contract's transfer mechanisms and allows for more reliable tracking and debugging of fee forwarding failures.

MU - Modifiers Usage

Criticality	Minor / Informative
Location	contracts/implementations/TokenImplementation.sol#L105,111,117
Status	Acknowledged

Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

```
require(newFee <= 1000, "Fee cannot exceed 10%");
...
require(newFee <= 500, "Fee cannot exceed 5%");
```

Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

Team Update

The team has acknowledged that this is not a security issue and states:

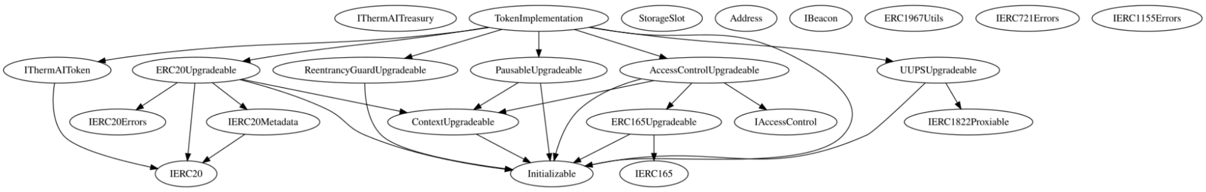
Will reconsider post-launch if needed.

Functions Analysis

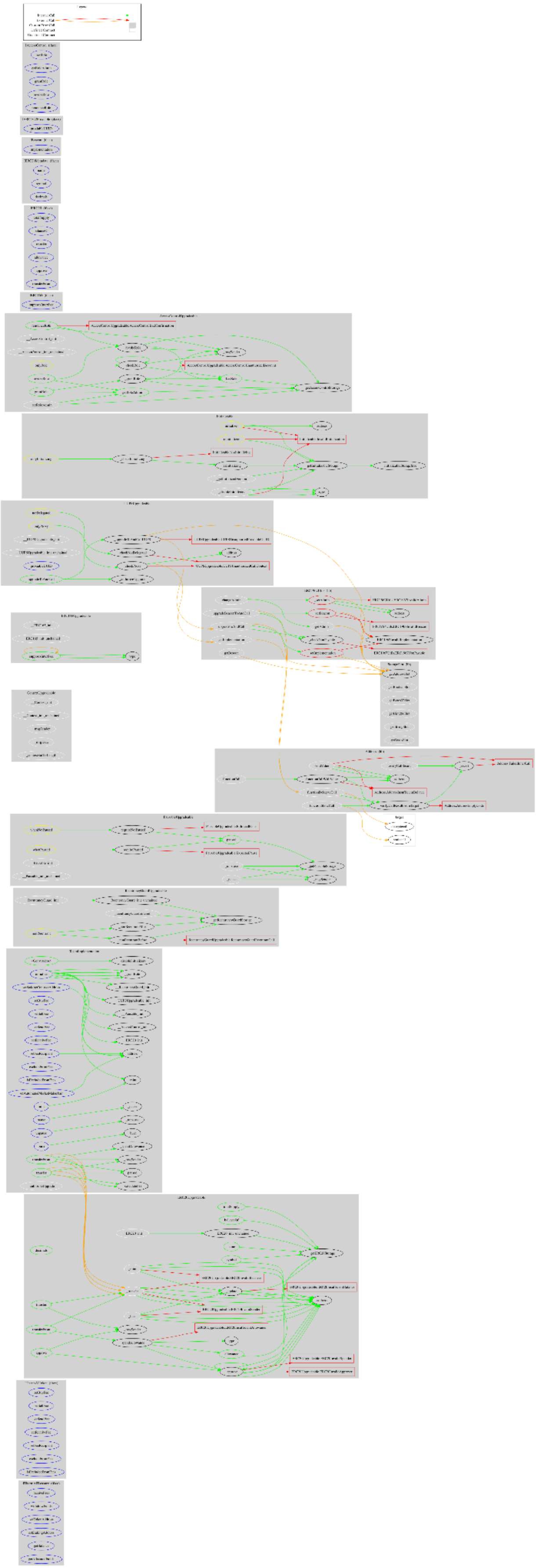
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
TokenImplementation	Implementation	Initializable, ERC20Upgradeable, AccessControlUpgradeable, PausableUpgradeable, UUPSUpgradeable, ReentrancyGuardUpgradeable, IThermAIToken		
		Public	✓	-
	initialize	External	✓	initializer
	setStakingContractAddress	External	✓	onlyRole
	setBuyFee	External	✓	onlyRole
	setSellFee	External	✓	onlyRole
	setSendFee	External	✓	onlyRole
	setReceiveFee	External	✓	onlyRole
	setFeeRecipient	External	✓	onlyRole
	excludeFromFees	External	✓	onlyRole
	isExcludedFromFees	External		-
	setAutomatedMarketMakerPair	External	✓	onlyRole

	_calculateFee	Internal		
	transfer	Public	✓	nonReentrant
	transferFrom	Public	✓	nonReentrant
	pause	External	✓	onlyRole
	unpause	External	✓	onlyRole
	mint	External	✓	onlyRole
	burn	External	✓	-
	_authorizeUpgrade	Internal	✓	onlyRole

Inheritance Graph



Flow Graph



Summary

ThermAI contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. There are some functions that can be abused by the owner like stop transactions and mint tokens. If the contract owner abuses the mint functionality, then the contract will be highly inflated. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a limit of max 10% fees.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an “endorsement” or “disapproval” of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

cyberscope.io