F Y E O

Security Code Review
Theoriq

Theoriq

July 2024
Version 1.0

Presented by:
FYEO Inc.
PO Box 147044
Lakewood CO 80214
United States

Security Level
Public

Table of Contents

Executive Summary....2
- Overview....2
- Key Findings....2
- Scope and Rules of Engagement....2

Technical Analyses and Findings....4
- Findings....5
- Technical Analysis....5
- Conclusion....5

Technical Findings....6
- General Observations....6
- Accounting does not work for tokens with transfer fees....7
- Bounds are not always checked....8
- Duplicate check....9
- Missing reentrancy check....10
- Missing zero address checks....11

Our Process....13
- Methodology....13
- Kickoff....13
- Ramp-up....13
- Review....14
- Code Safety....14
- Technical Specification Matching....14
- Reporting....15
- Verify....15

Additional Note....15

The Classification of vulnerabilities....16

Executive Summary

Overview

Theoriq engaged FYEO Inc. to perform a Security Code Review of the Theoriq Smart Contract.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on June 24 - July 01, 2024, and focused on the following objectives:

- To provide the customer with an assessment of their overall security posture and any risks that were discovered within the environment during the engagement.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the results of our tests.

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the FYEO Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

Key Findings

The following issues have been identified during the testing period. These should be prioritized for remediation to reduce the risk they pose:

- FYEO-THEORIQ-01 – Accounting does not work for tokens with transfer fees
- FYEO-THEORIQ-02 – Bounds are not always checked
- FYEO-THEORIQ-03 – Duplicate check
- FYEO-THEORIQ-04 – Missing reentrancy check
- FYEO-THEORIQ-05 – Missing zero address checks

Based on our review process, we conclude that the reviewed code implements the documented functionality.

Scope and Rules of Engagement

The FYEO Review Team performed a Security Code Review Theoriq. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

The source code was supplied through a private repository at https://github.com/chain-ml/theoriq-smart-contracts with the commit hash update-24-06.

Files included in the code review

theoriq-smart-contracts/
└── contracts/
  └── contracts/
  ├── AgentToken.sol
  ├── BaseContract.sol
  ├── CreditToken.sol
  ├── Escrow.sol
  ├── Proxy.sol
  └── Registry.sol

Table 1: Scope

Technical Analyses and Findings

During the Security Code Review Theoriq, we discovered:

- 1 finding with MEDIUM severity rating.
- 4 findings with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

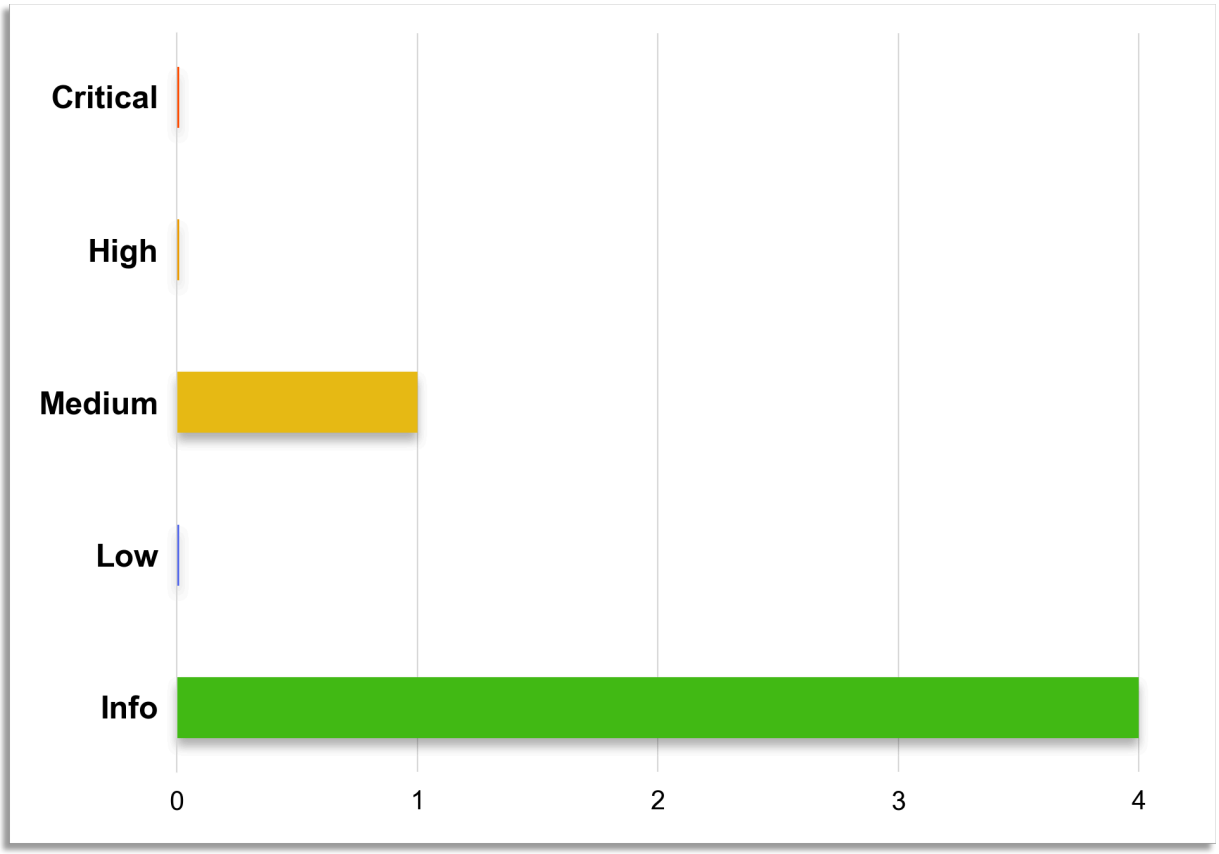

Figure 1: Findings by Severity

Findings

The *Findings* section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

Finding #	Severity	Description
FYEO-THEORIQ-01	**Medium**	Accounting does not work for tokens with transfer fees
FYEO-THEORIQ-02	**Informational**	Bounds are not always checked
FYEO-THEORIQ-03	**Informational**	Duplicate check
FYEO-THEORIQ-04	**Informational**	Missing reentrancy check
FYEO-THEORIQ-05	**Informational**	Missing zero address checks

Table 2: Findings Overview

Technical Analysis

The source code has been manually validated to the extent that the state of the repository allowed. The validation includes confirming that the code correctly implements the intended functionality.

Conclusion

Based on our review process, we conclude that the code implements the documented functionality to the extent of the reviewed code.

Technical Findings

General Observations

The Theoriq program utilizes smart contracts to facilitate transactions where users or AI agents pay other AI agents to perform jobs. The system operates on an EVM-compatible blockchain and includes four primary smart contracts: the registry, the agent and credit tokens and an escrow. All contracts can be updated, paused if needed, and have controlled access based on roles. The Registry contract is responsible for banning users, AgentToken signifies ownership of AI agents, CreditToken helps cover transaction costs, and Escrow handles the financial transactions for agent operations. Technically, any ERC20 token can be used for payments, but the admin maintains a whitelist to approve specific ones.

During the review, it was noted that the codebase is exceptionally well-crafted. The provided documentation is excellent, and the in-code comments are also very well done. Numerous tests confirm the correct operation of the smart contracts. Additionally, the team was very communicative and promptly addressed any questions that arose.

Accounting does not work for tokens with transfer fees

Finding ID: FYEO-THEORIQ-01
Severity: **Medium**
Status: **Remediated**

Description

The current accounting logic does not work correctly for tokens charging a transfer fee. This will result in the contract receiving less tokens than assumed.

Proof of Issue

File name: contracts/contracts/Escrow.sol
Line number: 207

```
_balances[currencyContractAddress][to].totalBalance += amount;
SafeERC20.safeTransferFrom(
    IERC20(currencyContractAddress),
    _msgSender(),
    address(this),
    amount
);
```

Severity and Impact Summary

Tokens that implement a transfer fee will consume this fee from the amount delivered to the recipient. Therefore the `amount` value will be higher than what was actually received by the contract.

Recommendation

Retrieve the contract's balance of the given ERC20 token before and after the `safeTransferFrom` operation. The amount actually received will be the difference between the two.

Bounds are not always checked

Finding ID: FYEO-THEORIQ-02
Severity: **Informational**
Status: **Open**

Description

Some of the variables remain without bound checks. The issue has been discussed and it was decided that this issue won't be addressed for now.

Proof of Issue

File name: contracts/contracts/CreditToken.sol
Line number: 151

```
if (maxUsesPerPeriod == 0) {
    revert InvalidArgument("maxUsesPerPeriod");
}
```

This does not establish an upper bound for `maxUsesPerPeriod`.

Line number: 165

```
_metadata[tokenId] = Credit({
    ...
    overallExpiryTimestamp: overallExpiryTimestamp,
    renewalPeriod: renewalPeriod,
```

These values will also accept numbers with 78 digits up to (2^256 -1).

Severity and Impact Summary

Not a security concern. It may become relevant when designing incentives around certain usage patterns. For instance if users were to receive incentives for using `overallExpiryTimestamp` with 0 (never expires) but some chose to use large numbers, they might miss out on these incentives.

Recommendation

Consider if upper bounds should be established.

Duplicate check

Finding ID: FYEO-THEORIQ-03
Severity: **Informational**
Status: **Remediated**

Description

There is a duplicate check if a currency is enabled.

Proof of Issue

File name: contracts/contracts/Escrow.sol
Line number: 182

```
function depositFunds(
    address to,
    uint256 amount,
    address currencyContractAddress
)
    ...
    currencyEnabled(currencyContractAddress)
{
    ...
    if (!_currencies[currencyContractAddress]) {
        revert CurrencyNotEnabled();
    }
```

Severity and Impact Summary

No security impact.

Recommendation

Remove the duplicate check.

Missing reentrancy check

Finding ID: FYEO-THEORIQ-04
Severity: **Informational**
Status: **Remediated**

Description

The agent token's `safeMint` function is missing a reentrancy check. In comparison, the credit token does implement this check.

Proof of Issue

File name: contracts/contracts/AgentToken.sol
Line number: 172

```
function safeMint(
    address to,
    uint256 tokenId,
    TokenParams memory params
) external onlyRole(MINTER_ROLE) whenNotPaused {
```

Severity and Impact Summary

This is an authorized function, but it does call onERC721Received.

Recommendation

Add a reentrancy guard.

Missing zero address checks

Finding ID: FYEO-THEORIQ-05
Severity: **Informational**
Status: **Remediated**

Description

There are a few instances where addresses should be checked against the zero address.

Proof of Issue

File name: contracts/contracts/BaseContract.sol
Line number: 41

```
function __BaseContract_init(
    address defaultAdmin,
    address pauser,
    address upgrader
) internal onlyInitializing {
    __Pausable_init();
    __AccessControl_init();
    __UUPSUpgradeable_init();

    _grantRole(DEFAULT_ADMIN_ROLE, defaultAdmin);
    _grantRole(PAUSER_ROLE, pauser);
    _grantRole(UPGRADER_ROLE, upgrader);
}
```

When calling `__BaseContract_init`, it should be checked that `defaultAdmin` is not zero.

File name: contracts/contracts/AgentToken.sol
Line number: 139

```
function initialize(
    address defaultAdmin,
    address pauser,
    address upgrader,
    address minter,
    address freezer,
    address registry
) public initializer {
    __BaseContract_init(defaultAdmin, pauser, upgrader);
    _grantRole(MINTER_ROLE, minter);
    _grantRole(FREEZER_ROLE, freezer);
    _registry = Registry(registry);
}
```

It should be checked that `_registry` is not zero. This also applies to the `CreditToken`.

File name: contracts/contracts/BaseContract.sol
Line number: 83

```
function grantRole(
    bytes32 role,
```

```
    address account
) public override whenNotPaused {
    AccessControlUpgradeable.grantRole(role, account);
}
```

The `account` can be the zero address.

Severity and Impact Summary

By mistake the contracts could be initialized with the wrong values.

Recommendation

Make sure to add checks against the zero address where required.

Our Process

Methodology

FYEO Inc. uses the following high-level methodology when approaching engagements. They are broken up into the following phases.

Figure 2: Methodology Flow

Kickoff

The project is kicked off as the sales process has concluded. We typically set up a kickoff meeting where project stakeholders are gathered to discuss the project as well as the responsibilities of participants. During this meeting we verify the scope of the engagement and discuss the project activities. It's an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there is an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

Ramp-up

Ramp-up consists of the activities necessary to gain proficiency on the project. This can include the steps needed for familiarity with the codebase or technological innovation utilized. This may include, but is not limited to:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for specific languages
- Researching common flaws and recent technological advancements

Review

The review phase is where most of the work on the engagement is completed. This is the phase where we analyze the project for flaws and issues that impact the security posture. Depending on the project this may include an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project
3. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following sections.

Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This list is general and not comprehensive, meant only to give an understanding of the issues we are looking for.

Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description

Reporting

FYEO Inc. delivers a draft report that contains an executive summary, technical details, and observations about the project.

The executive summary contains an overview of the engagement including the number of findings as well as a statement about our general risk assessment of the project. We may conclude that the overall risk is low but depending on what was assessed we may conclude that more scrutiny of the project is needed.

We report security issues identified, as well as informational findings for improvement, categorized by the following labels:

- Critical
- High
- Medium
- Low
- Informational

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

As we perform the audit, we may identify issues that aren't security related, but are general best practices and steps that can be taken to lower the attack surface of the project. We will call those out as we encounter them and as time permits.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

Verify

After the preliminary findings have been delivered, this could be in the form of the approved communication channel or delivery of the draft report, we will verify any fixes within a window of time specified in the project. After the fixes have been verified, we will change the status of the finding in the report from open to remediated.

The output of this phase will be a final report with any mitigated findings noted.

Additional Note

It is important to note that, although we did our best in our analysis, no code audit or assessment is a guarantee of the absence of flaws. Our effort was constrained by resource and time limits along with the scope of the agreement.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination.

The Classification of vulnerabilities

Security vulnerabilities and areas for improvement are weighted into one of several categories using, but is not limited to, the criteria listed below:

Critical – vulnerability will lead to a loss of protected assets

- This is a vulnerability that would lead to immediate loss of protected assets
- The complexity to exploit is low
- The probability of exploit is high

High - vulnerability has potential to lead to a loss of protected assets

- All discrepancies found where there is a security claim made in the documentation that cannot be found in the code
- All mismatches from the stated and actual functionality
- Unprotected key material
- Weak encryption of keys
- Badly generated key materials
- Txn signatures not verified
- Spending of funds through logic errors
- Calculation errors overflows and underflows

Medium - vulnerability hampers the uptime of the system or can lead to other problems

- Insecure calls to third party libraries
- Use of untested or nonstandard or non-peer-reviewed crypto functions
- Program crashes, leaves core dumps or writes sensitive data to log files

Low – vulnerability has a security impact but does not directly affect the protected assets

- Overly complex functions
- Unchecked return values from 3rd party libraries that could alter the execution flow

Informational

- General recommendations