# Token

## Smart Contract Audit Report
## Prepared for Zentry

| | |
|---|---|
| **Date Issued:** | Mar 29, 2024 |
| **Project ID:** | AUDIT2024005 |
| **Version:** | v1.0 |
| **Confidentiality Level:** | Public |

CYBERSECURITY PROFESSIONAL SERVICE

## Report Information

| | |
|---|---|
| **Project ID** | AUDIT2024005 |
| **Version** | v1.0 |
| **Client** | Zentry |
| **Project** | Token |
| **Auditor(s)** | Phitchakorn Apiratisakul<br>Wachirawit Kanpanluk |
| **Author(s)** | Phitchakorn Apiratisakul |
| **Reviewer** | Natsasit Jirathammanuwat |
| **Confidentiality Level** | Public |

## Version History

| Version | Date | Description | Author(s) |
|---|---|---|---|
| 1.0 | Mar 29, 2024 | Full report | Phitchakorn Apiratisakul |

## Contact Information


| | |
|---|---|
| **Company** | Inspex |
| **Phone** | (+66) 90 888 7186 |
| **Telegram** | t.me/inspexco |
| **Email** | audit@inspex.co |

# Table of Contents

# 1. Executive Summary

As requested by Zentry, Inspex team conducted an audit to verify the security posture of the Token smart contracts on Mar 27, 2024. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of Token smart contracts. Static code analysis, dynamic analysis, and manual review were done in conjunction to identify smart contract vulnerabilities together with technical & business logic flaws that may be exposed to the potential risk of the platform and the ecosystem. Practical recommendations are provided according to each vulnerability found and should be followed to remediate the issue.

## 1.1. Audit Result

In the initial audit, Inspex found 1 very low, and 1 info-severity issues. With the project team's prompt response, 1 very low and 1 info-severity issues were acknowledged by the team. Therefore, Inspex trusts that Token smart contracts have sufficient protections to be safe for public use. However, in the long run, Inspex suggests resolving all issues found in this report.


## 1.2. Disclaimer

This security audit is not produced to supplant any other type of assessment and does not guarantee the discovery of all security vulnerabilities within the scope of the assessment. However, we warrant that this audit is conducted with goodwill, professional approach, and competence. Since an assessment from one single party cannot be confirmed to cover all possible issues within the smart contract(s), Inspex suggests conducting multiple independent assessments to minimize the risks. Lastly, nothing contained in this audit report should be considered as investment advice.

# 2. Project Overview

## 2.1. Project Introduction

The ZentryToken contract is an ERC20 token that users can migrate from the GuildFi tokens to the Zentry tokens using the Migrator contract. The migration feature can be enabled or disabled by the contract owner.

**Scope Information:**

| | |
|---|---|
| **Project Name** | Token |
| **Website** | https://zentry.com/ |
| **Smart Contract Type** | Ethereum Smart Contract |
| **Chain** | Ethereum |
| **Programming Language** | Solidity |
| **Category** | Token |

**Audit Information:**

| | |
|---|---|
| **Audit Method** | Whitebox |
| **Audit Date** | Mar 27, 2024 |
| **Reassessment Date** | Mar 29, 2024 |

The audit method can be categorized into two types depending on the assessment targets provided:

1. **Whitebox**: The complete source code of the smart contracts are provided for the assessment.
2. **Blackbox**: Only the bytecodes of the smart contracts are provided for the assessment.

## 2.2. Scope

The following smart contracts were audited and reassessed by Inspex in detail:

**Initial Audit:**

| Contract | Location (URL) |
|---|---|
| ZentryToken | https://etherscan.io/address/0xdBB7a34Bf10169d6d2D0d02A6cbb436cF4381BFa |
| Migrator | https://etherscan.io/address/0x9f28c9c2da4a833cbfaaacbf7eb62267334d7149 |

**Reassessment:**

| Contract | Location (URL) |
|---|---|
| ZentryToken | https://etherscan.io/address/0xdBB7a34Bf10169d6d2D0d02A6cbb436cF4381BFa |
| Migrator | https://etherscan.io/address/0x9f28c9c2da4a833cbfaaacbf7eb62267334d7149 |

The assessment scope covers only the in-scope smart contracts and the smart contracts that they inherit from.

## 2.3. Security Model

### 2.3.1 Trust Modules

The Zentry token has a privileged role with the authority to mutate the critical state variables of the contract. Changes to these state variables significantly impact the contract's functionality. The privileged roles and their corresponding privileged functions are enumerated as follows:

- The owner address in the migrator contract can enable or disable the `migrate()` function, this ability could allow the owner to prevent the migration process at any moment.

### 2.3.2 Trust Assumptions

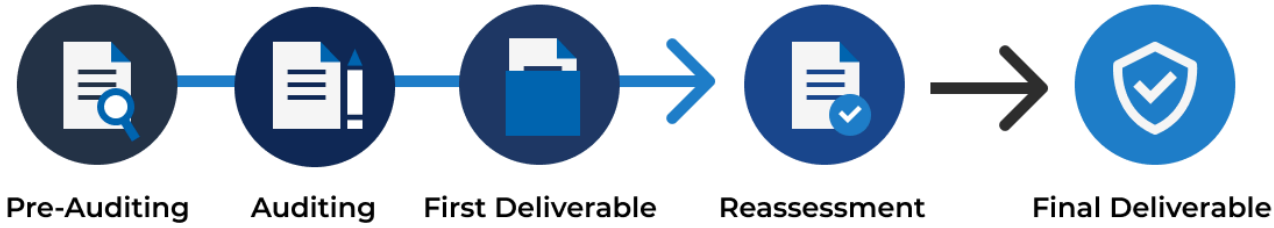
In the Zentry token, the protocol's privileged role, has the ability to change the critical state variables of the contract, also external components were assumed to be trusted. Acknowledging these trust assumptions is important, as it introduces substantial risks to the platform. Trust assumptions include, but are not limited to:

- All privileged roles perform the privileged function with good will
- The owner address is trusted to enable or disable the migration process for the users.

# 3. Methodology

Inspex conducts the following procedure to enhance the security level of our clients' smart contracts:

1. **Pre-Auditing**: Getting to understand the overall operations of the related smart contracts, checking for readiness, and preparing for the auditing
2. **Auditing**: Inspecting the smart contracts using automated analysis tools and manual analysis by a team of professionals
3. **First Deliverable and Consulting**: Delivering a preliminary report on the findings with suggestions on how to remediate those issues and providing consultation
4. **Reassessment**: Verifying the status of the issues and whether there are any other complications in the fixes applied
5. **Final Deliverable**: Providing a full report with the detailed status of each issue




## 3.1. Test Categories

Inspex smart contract auditing methodology consists of both automated testing with scanning tools and manual testing by experienced testers. We have categorized the tests into 3 categories as follows:

1. **General Smart Contract Vulnerability (General)** - Smart contracts are analyzed automatically using static code analysis tools for general smart contract coding bugs, which are then verified manually to remove all false positives generated.
2. **Advanced Smart Contract Vulnerability (Advanced)** - The workflow, logic, and the actual behavior of the smart contracts are manually analyzed in-depth to determine any flaws that can cause technical or business damage to the smart contracts or the users of the smart contracts.
3. **Smart Contract Best Practice (Best Practice)** - The code of smart contracts is then analyzed from the development perspective, providing suggestions to improve the overall code quality using standardized best practices.

## 3.2. Audit Items

The testing items checked are based on our Smart Contract Security Testing Guide (SCSTG) v1.0 (https://github.com/InspexCo/SCSTG/releases/download/v1.0/SCSTG_v1.0.pdf) which covers most prevalent risks in smart contracts. The latest version of the document can also be found at (https://docs.inspex.co/smart-contract-security-testing-guide/).

The following audit items were checked during the auditing activity:

| Testing Category | Testing Items |
|---|---|
| 1. Architecture and Design | 1.1. Proper measures should be used to control the modifications of smart contract logic<br>1.2. The latest stable compiler version should be used<br>1.3. The circuit breaker mechanism should not prevent users from withdrawing their funds<br>1.4. The smart contract source code should be publicly available<br>1.5. State variables should not be unfairly controlled by privileged accounts<br>1.6. Least privilege principle should be used for the rights of each role |
| 2. Access Control | 2.1. Contract self-destruct should not be done by unauthorized actors<br>2.2. Contract ownership should not be modifiable by unauthorized actors<br>2.3. Access control should be defined and enforced for each actor roles<br>2.4. Authentication measures must be able to correctly identify the user<br>2.5. Smart contract initialization should be done only once by an authorized party<br>2.6. tx.origin should not be used for authorization |
| 3. Error Handling and Logging | 3.1. Function return values should be checked to handle different results<br>3.2. Privileged functions or modifications of critical states should be logged<br>3.3. Modifier should not skip function execution without reverting |
| 4. Business Logic | 4.1. The business logic implementation should correspond to the business design<br>4.2. Measures should be implemented to prevent undesired effects from the ordering of transactions<br>4.3. msg.value should not be used in loop iteration |
| 5. Blockchain Data | 5.1. Result from random value generation should not be predictable<br>5.2. Spot price should not be used as a data source for price oracles<br>5.3. Timestamp should not be used to execute critical functions<br>5.4. Plain sensitive data should not be stored on-chain<br>5.5. Modification of array state should not be done by value<br>5.6. State variable should not be used without being initialized |

| Testing Category | Testing Items |
|---|---|
| 6. External Components | 6.1. Unknown external components should not be invoked<br>6.2. Funds should not be approved or transferred to unknown accounts<br>6.3. Reentrant calling should not negatively affect the contract states<br>6.4. Vulnerable or outdated components should not be used in the smart contract<br>6.5. Deprecated components that have no longer been supported should not be used in the smart contract<br>6.6. Delegatecall should not be used on untrusted contracts |
| 7. Arithmetic | 7.1. Values should be checked before performing arithmetic operations to prevent overflows and underflows<br>7.2. Explicit conversion of types should be checked to prevent unexpected results<br>7.3. Integer division should not be done before multiplication to prevent loss of precision |
| 8. Denial of Services | 8.1. State changing functions that loop over unbounded data structures should not be used<br>8.2. Unexpected revert should not make the whole smart contract unusable<br>8.3. Strict equalities should not cause the function to be unusable |
| 9. Best Practices | 9.1. State and function visibility should be explicitly labeled<br>9.2. Token implementation should comply with the standard specification<br>9.3. Floating pragma version should not be used<br>9.4. Builtin symbols should not be shadowed<br>9.5. Functions that are never called internally should not have public visibility<br>9.6. Assert statement should not be used for validating common conditions |

## 3.3. Risk Rating

OWASP Risk Rating Methodology (https://owasp.org/www-community/OWASP_Risk_Rating_Methodology) is used to determine the severity of each issue with the following criteria:

- **Likelihood**: a measure of how likely this vulnerability is to be uncovered and exploited by an attacker
- **Impact**: a measure of the damage caused by a successful attack

Both likelihood and impact can be categorized into three levels: **Low**, **Medium**, and **High**.

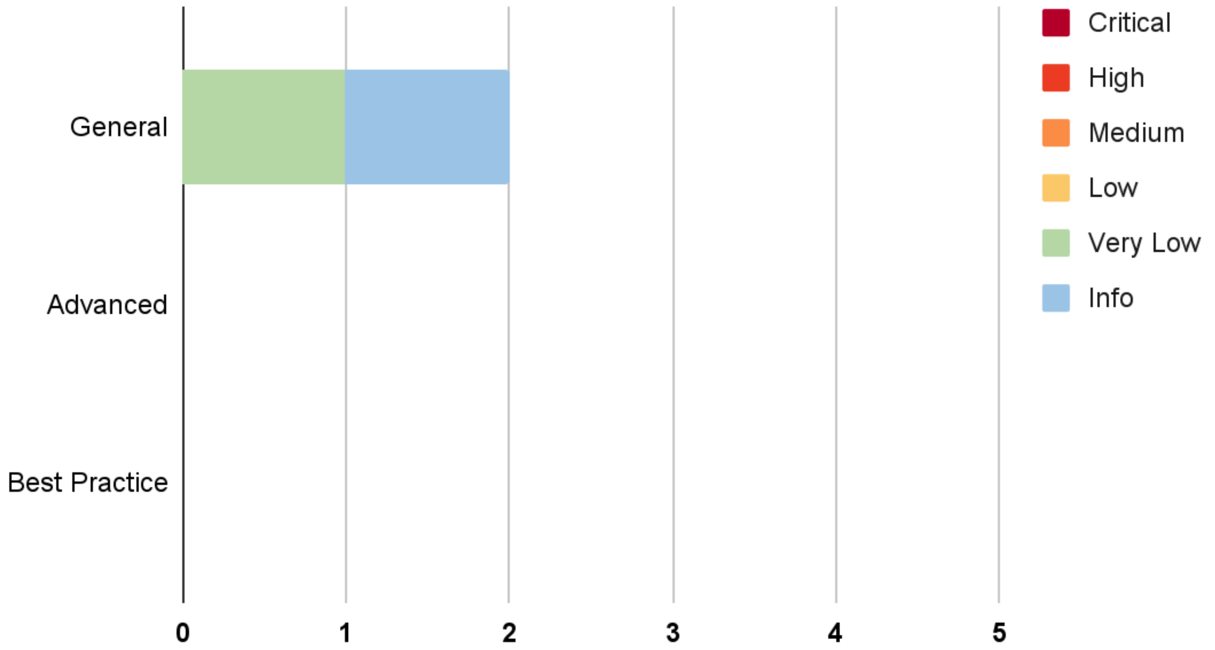
**Severity** is the overall risk of the issue. It can be categorized into five levels: **Very Low**, **Low**, **Medium**, **High**, and **Critical**. It is calculated from the combination of likelihood and impact factors using the matrix below. The severity of findings with no likelihood or impact would be categorized as **Info**.

| Likelihood<br>Impact | Low | Medium | High |
|---|---|---|---|
| Low | Very Low | Low | Medium |
| Medium | Low | Medium | High |
| High | Medium | High | Critical |

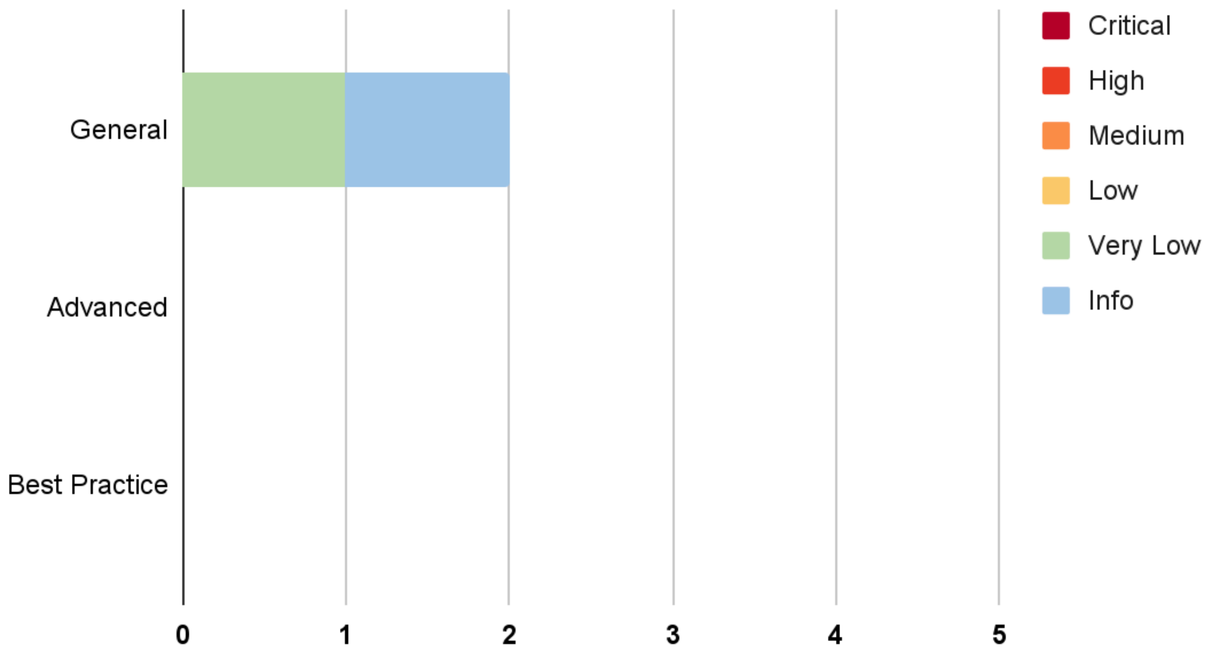
# 4. Summary of Findings

The following charts show the number of the issues found during the assessment and the issues acknowledged in the reassessment, categorized into three categories: **General**, **Advanced**, and **Best Practice**.

**Assessment:**




**Reassessment:**



The statuses of the issues are defined as follows:

| Status | Description |
|---|---|
| Resolved | The issue has been resolved and has no further complications. |
| Resolved * | The issue has been resolved with mitigations and clarifications. For the clarification or mitigation detail, please refer to Chapter 5. |
| Acknowledged | The issue's risk has been acknowledged and accepted. |
| No Security Impact | The best practice recommendation has been acknowledged. |

The information and status of each issue can be found in the following table:

| ID | Title | Category | Severity | Status |
|---|---|---|---|---|
| IDX-001 | Insufficient Logging for Privileged Functions | General | Very Low | Acknowledged |
| IDX-002 | Outdated Compiler Version | General | Info | No Security Impact |

* The mitigations or clarifications by Zentry can be found in Chapter 5.

# 5. Detailed Findings Information

## 5.1. Insufficient Logging for Privileged Functions

| | |
|---|---|
| **ID** | IDX-001 |
| **Target** | Migrator |
| **Category** | General Smart Contract Vulnerability |
| **CWE** | CWE-778: Insufficient Logging |
| **Risk** | **Severity:** Very Low<br><br>**Impact:** Low<br>Privileged functions' executions cannot be monitored easily by the users.<br><br>**Likelihood:** Low<br>It is not likely that the execution of the privileged functions will be a malicious action. |
| **Status** | **Acknowledged**<br>The Zentry team acknowledged this issue since there was no critical impact on the platform and the transaction was also publicly available on the blockchain. |

### 5.1.1. Description

Privileged functions that are executable by the controlling parties are not logged properly by emitting events. Without events, it is not easy for the public to monitor the execution of those privileged functions, allowing the controlling parties to perform actions that cause big impacts on the platform.

For example, the owner can enable migration by executing the `enableMigration()` function in the `Migrator` contract, and no events are emitted.

The privileged functions without sufficient logging are as follows:

| File | Contract | Function |
|---|---|---|
| Migrator.sol (L: 26) | Migrator | enableMigration() |
| Migrator.sol (L: 30) | Migrator | disableMigration() |

### 5.1.2. Remediation

Inspex suggests emitting events for the execution of privileged functions, for example:

**Migrator.sol**

```
event MigrationEnabled(bool enabled);

```

```
function enableMigration() external onlyOwner {
    migrationEnabled = true;
    emit MigrationEnabled(true);
}
```

## 5.2. Outdated Compiler Version

| | |
|---|---|
| **ID** | IDX-002 |
| **Target** | ZentryToken<br>Migrator |
| **Category** | General Smart Contract Vulnerability |
| **CWE** | CWE-1104: Use of Unmaintained Third Party Components |
| **Risk** | **Severity:** Info<br><br>**Impact: None**<br><br>**Likelihood: None** |
| **Status** | No Security Impact<br>The Zentry team acknowledged this issue since there is no security impact on the platform. |

### 5.2.1. Description

The Solidity compiler versions specified in the smart contracts were outdated (https://docs.soliditylang.org/en/latest/bugs.html). As the compilers are regularly updated with bug fixes and new features, the latest stable compiler version should be used to compile the smart contracts for best practice.

**ZentryToken.sol**

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.20;
```

**Migrator.sol**

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.20;
```

### 5.2.2. Remediation

Inspex suggests upgrading the Solidity compiler to the latest stable version (https://github.com/ethereum/solidity/releases). At the time of audit, the latest stable version of the Solidity compiler in major 0.8 is 0.8.25.

**ZentryToken.sol**

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.25;
```

Migrator.sol

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.25;
```

# 6. Appendix

## 6.1. About Inspex

Inspex is formed by a team of cybersecurity experts highly experienced in various fields of cybersecurity. We provide blockchain and smart contract professional services at the highest quality to enhance the security of our clients and the overall blockchain ecosystem.

**Follow Us On:**

| | |
|---|---|
| **Website** | https://inspex.co |
| **Twitter** | @InspexCo |
| **Facebook** | https://www.facebook.com/InspexCo |
| **Telegram** | @inspex_announcement |

inspex

CYBERSECURITY PROFESSIONAL SERVICE