Staking Pool V2

Smart Contract Audit Report
Prepared for deFusion

[deFusion]

Date Issued:	Mar 11, 2024
Project ID:	AUDIT2023004
Version:	v1.0
Confidentiality Level:	Public

inspeX
CYBERSECURITY PROFESSIONAL SERVICE

Report Information

Project ID	AUDIT2023004
Version	v1.0
Client	deFusion
Project	Staking Pool V2
Auditor(s)	Natsasit Jirathammanuwat
Author(s)	Natsasit Jirathammanuwat
Reviewer	Patipon Suwanbol
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
1.0	Mar 11, 2024	Full report	Natsasit Jirathammanuwat

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 deFusion, Inspex team conducted an audit to verify the security posture of the Staking Pool V2 smart contracts on Mar 5, 2024. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of Staking Pool V2 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 high, 1 medium, and 1 very low issues. With the project team's prompt response in resolving the issues found by Inspex, all issues were resolved in the reassessment. Therefore, Inspex trusts that Staking Pool V2 smart contracts have high-level protections in place to be safe from most attacks. However, as the source code is currently not publicly available, there is a potential risk that the smart contracts deployed on the blockchain may not be identical to the audited smart contracts. This discrepancy could result in introducing security vulnerabilities or unintended behaviors that were not identified during the audit process. It is of importance to recognize that interacting with an unverified smart contract may lead to the potential loss of funds. In this case, the hash of the deployed smart contract bytecode should be compared with the hash of the audited smart contract bytecode to ensure that the deployed smart contract is identical to the audited smart contract before interacting with them. 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 deFusion staking pool is designed exclusively for whales seeking a secure and reliable B2B staking experience. With cutting-edge technology and a commitment to transparency, deFusion provides a trustworthy platform for users to stake their tokens, ensuring a fusion of stability and strength in the ever-evolving blockchain landscape.

Scope Information:

Project Name	Staking Pool V2
Website	https://www.defusion.xyz/
Smart Contract Type	Ethereum Smart Contract
Chain	Viction
Programming Language	Solidity
Category	Staking Pool

Audit Information:

Audit Method	Whitebox
Audit Date	Mar 5, 2024 - Mar 5, 2024
Reassessment Date	Mar 8, 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 smart contracts with the following bytecodes were audited and reassessed by Inspex in detail:

Initial Audit:

Contract	Bytecode SHA256 Hash
BroadcasterHelper	-
DefusionFactory	3852a61446a23e59a7efaa536955b92f92aa1c6e84868cd0e84aaa39dae8d23b
Vpool	9d8d973bd368047b3dbdcc3ff4d3ddfb5ab35b37ed09ef3c8a7498353c66f243
VRC25	-

Reassessment Audit:

Contract	Bytecode SHA256 Hash
BroadcasterHelper	-
DefusionFactory	aa3a05f78dacff86271d7c456a7b397736a17e672ed1c0059d11cb08afe762e1
Vpool	9d8d973bd368047b3dbdcc3ff4d3ddfb5ab35b37ed09ef3c8a7498353c66f243
VRC25	-

compiler_config.json

```
{
    solidity: {
        version: "0.8.16",
        settings: {
            optimizer: {
                enabled: true,
                    runs: 200
            }
        }
    }
}
```

As the deFusion team has decided not to publish the source code to protect their intellectual property, the users should compare the bytecode hashes with the smart contracts deployed before interacting with them to make sure that they are the same with the contracts audited.

2.3. Security Model

2.3.1 Trust Modules

The deFusion has privileged roles 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 role and their corresponding privileged function are enumerated as follows:

- The owner address can perform the following actions:
  - Set the pool implementation by executing the `setImplement()` function. Changing the pool implementation could possibly result in unauthorized minting or burning of the `DefusionFactory` tokens; this can be maliciously used to permanently lock users' staked funds.
  - Set the protocol fee freely from 0 up to nearly 99.99% with the `setProtocolFee()` function.
  - Set the protocol fee recipient address, which may lead to denial of service if the fee transfer is reverted.
  - Grant and revoke the admin role to any address.
- The admin address can perform the following actions:
  - Create a new staking pool with the `createPool()` function.
  - Apply VRC25 to each staking pool with the `applyVRC25()` function.

The deFusion several functionalities have relied on the external component, which may significantly impact the contract if they malfunction. The external components are listed as follows:

- The `TomoValidator` contract offers a range of functionalities related to the validator staking reward, spanning from proposing and voting to unvoting, resigning, and withdrawing.
- The `TRC25Issuer` contract offers issuing functionalities for gas-sponsored token transfers.
- The `BroadcasterHelper` contract is presumed to emit events without encountering transaction reverts.

2.3.2 Trust Assumptions

In the deFusion, the protocol's privileged roles, have the ability to change the critical state variable of the contract, also external components such as `TomoValidator` contract 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:

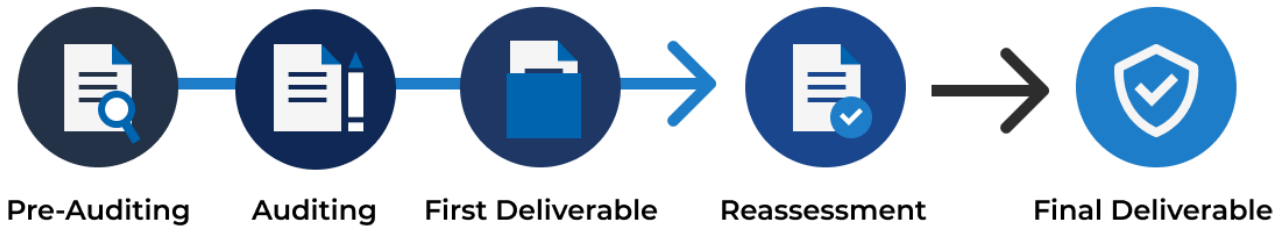
- All privileged addresses perform the privileged function with good will.
- The owner address is entrusted to change the pool contract implementation to only legitimate contracts.
- The owner address is entrusted to change the protocol fee according to the protocol documentation.
- The owner address is entrusted to change the protocol fee recipient address correctly.
- The admin address is entrusted to create the legitimate staking pool.
- The admin address is entrusted to apply VRC25 correctly.

- It is presumed that the `TomoValidator` contract consistently functions as intended.
- It is presumed that the `TRC25Issuer` contract consistently issues the VRC25 as intended.
- It is presumed that the `Broadcaster` contract consistently emits events correctly as intended.

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 1.2. The latest stable compiler version should be used 1.3. The circuit breaker mechanism should not prevent users from withdrawing their funds 1.4. The smart contract source code should be publicly available 1.5. State variables should not be unfairly controlled by privileged accounts 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 2.2. Contract ownership should not be modifiable by unauthorized actors 2.3. Access control should be defined and enforced for each actor roles 2.4. Authentication measures must be able to correctly identify the user 2.5. Smart contract initialization should be done only once by an authorized party 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 3.2. Privileged functions or modifications of critical states should be logged 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 4.2. Measures should be implemented to prevent undesired effects from the ordering of transactions 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 5.2. Spot price should not be used as a data source for price oracles 5.3. Timestamp should not be used to execute critical functions 5.4. Plain sensitive data should not be stored on-chain 5.5. Modification of array state should not be done by value 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 6.2. Funds should not be approved or transferred to unknown accounts 6.3. Reentrant calling should not negatively affect the contract states 6.4. Vulnerable or outdated components should not be used in the smart contract 6.5. Deprecated components that have no longer been supported should not be used in the smart contract 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 7.2. Explicit conversion of types should be checked to prevent unexpected results 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 8.2. Unexpected revert should not make the whole smart contract unusable 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 9.2. Token implementation should comply with the standard specification 9.3. Floating pragma version should not be used 9.4. Builtin symbols should not be shadowed 9.5. Functions that are never called internally should not have public visibility 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**.

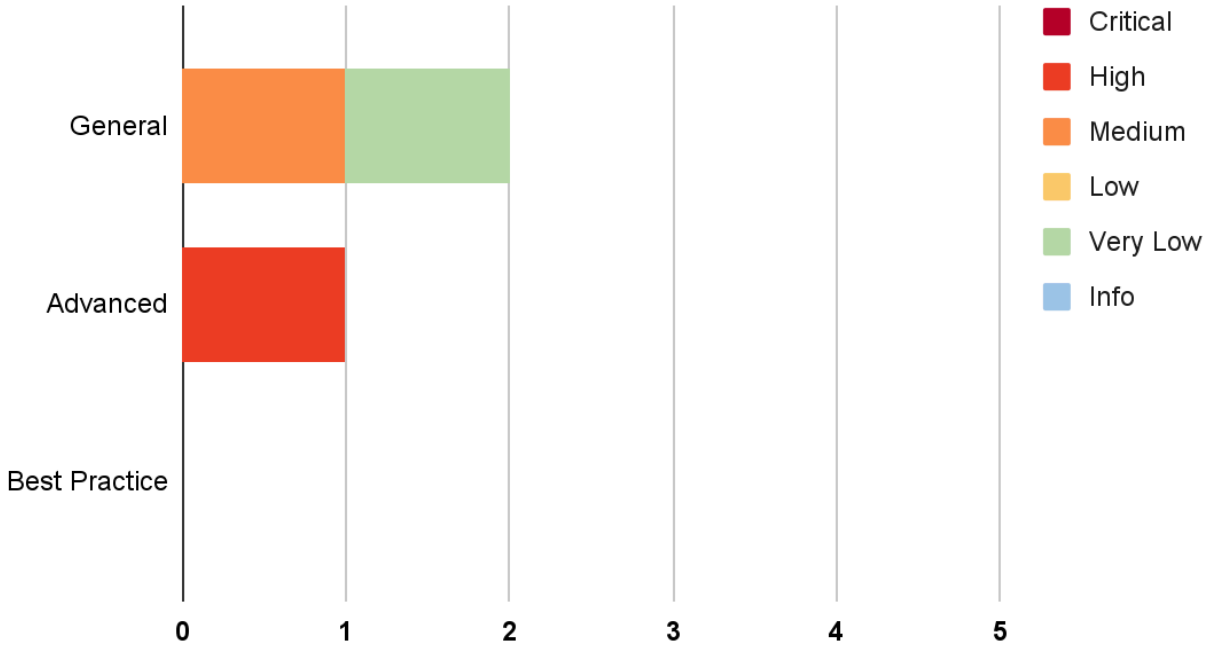
Severity is the overall risk of the issue. It can be categorized into five levels: **Very Low**, **Low**, **Medium**, **High**, and **Critical**. It is calculated from the combination of likelihood and impact factors using the matrix below. The severity of findings with no likelihood or impact would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

Impact \ Likelihood	Low	Medium	High
Low	Very Low	Low	Medium
Medium	Low	Medium	High
High	Medium	High	Critical

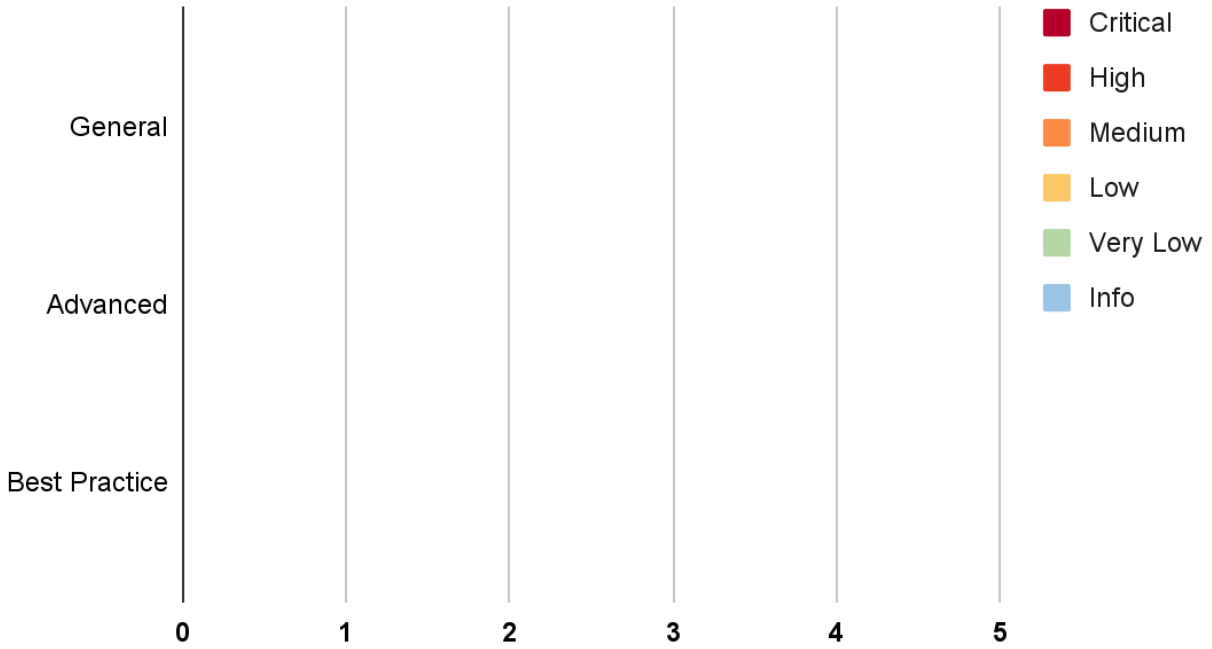
4. Summary of Findings

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	Incorrect Gas Sponsored Fee Collection	Advanced	High	Resolved
IDX-002	Centralized Authority Control	General	Medium	Resolved
IDX-003	Insufficient Logging for Privileged Functions	General	Very Low	Resolved

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

5. Detailed Findings Information

5.1. Incorrect Gas Sponsored Fee Collection

ID	IDX-001
Target	VRC25
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** High **Impact:** High The LP token will be extra charged from the user to the owner due to gas-sponsored fee logic, which does not comply with the business requirement. Furthermore, the collected LP token also cannot be unstaked by the owner, resulting in funds being permanently locked in the contract. **Likelihood:** Medium This issue arises when the user transfers the LP token by directly calling the contract, which triggers the gas-sponsored fee logic.
Status	Resolved The deFusion team has resolved this issue by overriding the `estimateFee()` to return 0 fee in the `DefusionFactory` contract.

5.1.1. Description

The `DefusionFactory` contract is the LP token of the platform, which inherits from the `VRC25` contract for supporting the gas-sponsored feature, as shown below:

Factory.sol

```
18 contract DefusionFactory is VRC25, ProxyFactory, TimeLock, BroadcasterHelper {
19     using SafeMath for uint256;
```

While transferring the `VRC25` token, the gas-sponsored fee will be extra charged from the user's wallet and transferred to the owner. The `estimateFee()` function is used to calculate the fee amount to be charged, as shown in the `transfer()` and `transferFrom()` functions below:

VRC25.sol

```
130   function transfer(address recipient, uint256 amount) external override
    returns (bool) {
131     uint256 fee = estimateFee(amount);
132     _transfer(msg.sender, recipient, amount);
133     _chargeFeeFrom(msg.sender, recipient, fee);
```

```
134     return true;
135   }
```

VRC25.sol

```
167   function transferFrom(address sender, address recipient, uint256 amount)
external override returns (bool) {
168     uint256 fee = estimateFee(amount);
169     require(_allowances[sender][msg.sender] >= amount.add(fee), "VRC25: amount
exeeds allowance");
170
171     _allowances[sender][msg.sender] =
_allowances[sender][msg.sender].sub(amount).sub(fee);
172     _transfer(sender, recipient, amount);
173     _chargeFeeFrom(sender, recipient, fee);
174     return true;
175   }
```

Since the `_minFee` has never been set, it remains the default value, which is 0. Therefore, the fee calculated from the `estimateFee()` function will be the same as the amount of token transferred.

VRC25.sol

```
113 function estimateFee(uint256 value) public virtual view override returns
(uint256) {
114   if(address(msg.sender).isContract()) {
115     return 0;
116   }
117   if(value > _minFee) {
118     return value;
119   }
120   return _minFee;
121 }
```

This results in a very large number of additional fees being charged to the user, which conflicts with the business requirement. Furthermore, the owner is also unable to unstake the fund with the user's LP token since there is no owner's share accounting in the `Vpool` contract.

5.1.2. Remediation

Inspex suggests implementing the `estimateFee()` function to consistently return a value of 0, ensuring compliance with the business requirement that no fee should be collected during transfers.

VRC25.sol

```
113 function estimateFee(uint256 value) public virtual view override returns
(uint256) {
114   return 0;
```

```
115 }
```

5.2. Centralized Authority Control

ID	IDX-002
Target	DefusionFactory
Category	General Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity:** Medium **Impact:** High The controlling authorities have the ability to manipulate critical state variables, altering the contract's behavior and potentially increasing their profits. This manipulation can lead to malfunctions and render the system unreliable, which is unfair to platform users. **Likelihood:** Low The private key of the controlling authorities is unlikely to be compromised. Nevertheless, if it were to be compromised, there are no restrictions preventing unauthorized changes from being made.
Status	Resolved The deFusion team has resolved this issue by adding the `whenUnlock` modifier which applies a time delay mechanism for the affected functions.

5.2.1. Description

Critical state variables can be updated at any time by the controlling authorities. Changes in these variables can cause impacts to the users, so the users should accept or be notified before these changes are effective.

However, there is currently no mechanism in place to prevent the authorities from altering these variables without notifying the users. In the event that the private key of the controlling authorities is compromised by an attacker, they can manipulate the contract's behavior and profit without any restrictions.

The following table contains all functions that modify critical state variables, potentially impacting the user platform.

File	Contract	Function
Factory.sol (L: 111)	DefusionFactory	setProtocolFee()
Factory.sol (L: 116)	DefusionFactory	setProtocolFeeAddress()

5.2.2. Remediation

In the ideal case, the critical state variables should not be modifiable to keep the integrity of the smart contract. However, if modification is needed, Inspex suggests implementing a community-run smart contract

governance to control the use of this function.

If removing the function or implementing the smart contract governance is not possible, Inspex suggests mitigating the risk of this issue by using a timelock mechanism to delay the changes for a reasonable amount of time or using the multi-signature contract to reduce the risk of unauthorized or malicious alterations to the smart contract.

5.3. Insufficient Logging for Privileged Functions

ID	IDX-003
Target	DefusionFactory
Category	General Smart Contract Vulnerability
CWE	CWE-778: Insufficient Logging
Risk	**Severity:** Very Low **Impact:** Low Users cannot easily monitor the execution of privileged functions. **Likelihood:** Low The execution of privileged functions is unlikely to be a malicious action.
Status	Resolved The deFusion team has resolved this issue by adding the `_emitEvent()` function to emit the event.

5.3.1. 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 set the protocol fee of the staking pool by executing the `setProtocolFee()` function in the `DefusionFactory` contract, and no events are emitted.

Factory.sol

```
111 function setProtocolFee(uint256 _newProtocolFee) external onlyOwner {
112     require(_newProtocolFee > 0 && _newProtocolFee < 100_000, "Factory: invalid
    protocol fee");
113     _protocolFee = _newProtocolFee;
114 }
```

The following table contains all functions that do not emit an event.

File	Contract	Function
Factory.sol (L: 111)	DefusionFactory	setProtocolFee()
Factory.sol (L: 116)	DefusionFactory	setProtocolFeeAddress()

5.3.2. Remediation

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

Factory.sol

```
111 function setProtocolFee(uint256 _newProtocolFee) external onlyOwner {
112     require(_newProtocolFee > 0 && _newProtocolFee < 100_000, "Factory: invalid
    protocol fee");
113     _protocolFee = _newProtocolFee;
114     _emitEvent("SetProtocolFee", msg.sender, address(this),
    abi.encode(_newProtocolFee));
115 }
```

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