STRUCT

SMART CONTRACT AUDIT

zokyo

May 24th 2023 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

SCORE

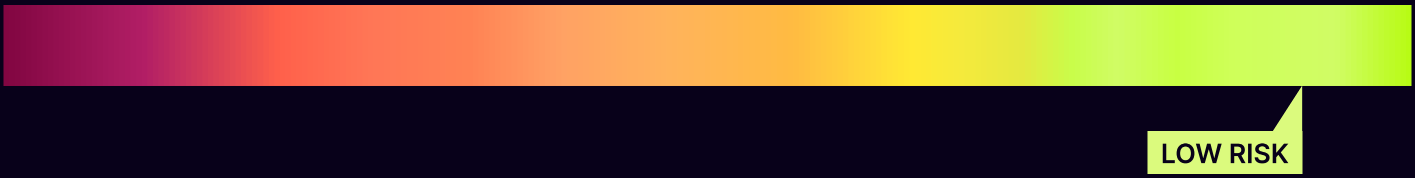
97

TECHNICAL SUMMARY

This document outlines the overall security of the Struct Finance smart contracts evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the Struct Finance smart contracts codebase for quality, security, and correctness.

Contract Status



There were 0 critical issues found during the audit. (See Complete Analysis)

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contracts but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the Struct Finance team put in place a bug bounty program to encourage further active analysis of the smart contracts.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Struct Finance repository:
https://github.com/struct-defi/struct-core/tree/audit/zokyo-feb-2023

Last commit: c7785ea69d5bfa00887e0be074606f7144aa5ac3

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- protocol/products/traderjoe/FEYTraderJoeProduct.sol
- protocol/products/traderjoe/FEYTraderJoeProductFactory.sol
- protocol/yield-sources/TraderJoeYieldSource.sol
- protocol/common/GACManaged.sol
- protocol/tokenization/StructERC1155.sol
- protocol/tokenization/StructSPToken.sol
- libraries/helpers/Constants.sol
- libraries/helpers/Errors.sol
- libraries/helpers/Helpers.sol
- libraries/logic/Validation.sol
- libraries/types/DataTypes.sols

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most resent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Struct Finance smart contracts. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01 Due diligence in assessing the overall code quality of the codebase.

02 Cross-comparison with other, similar smart contracts by industry leaders.

03 Thorough manual review of the codebase line by line.

Executive Summary

There was no critical issue found during the audit. Our auditing team was discovered issues with medium and low severity, and couple of informational issues. All the mentioned findings may have an effect only in the case of specific conditions performed by the contract owner and the investors interacting with it. They are described in detail in the “Complete Analysis” section.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the Struct Finance team and the Struct Finance team is aware of it, but they have chosen to not solve it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

● **Critical**

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

● **High**

The issue affects the ability of the contract to compile or operate in a significant way.

● **Medium**

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

● **Low**

The issue has minimal impact on the contract’s ability to operate.

● **Informational**

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

FINDINGS SUMMARY

#	Title	Risk	Status
1	Centralization of emergencyRemoveLiquidity()	Medium	Resolved
2	Centralization of Pausable and possible Denial of Service	Medium	Resolved
3	Tokens can get stuck while calling deposit()	Medium	Resolved
4	Centralization of setFEYProductImplementation()	Medium	Resolved
5	Exposure to revert due to wrong operation.	Medium	Resolved
6	Authorization of only EOAs can break multisigs	Low	Resolved
7	Bypassing require checks possible	Low	Acknowledged
8	Missing zero address checks	Low	Acknowledged
9	Investors can lose funds due to delay in withdrawal	Low	Acknowledged
10	Misleading value assignment to poolId	Low	Acknowledged
11	Missing Input Validation in StructPriceOracle	Low	Acknowledged
12	Iterating allProducts array can cause the gas limit to exceed	Low	Unresolved
13	trancheDurationMax can be less than trancheDurationMin	Low	Resolved

#	Title	Risk	Status
16	Fees can be set greater than 100%	Low	Acknowledged
17	No check for zero amount	Low	Acknowledged
18	Gas Optimization	Informational	Unresolved
19	initialize() can be called by anyone	Informational	Acknowledged
20	Inverted naming of variables	Informational	Acknowledged
21	Critical function `rescueTokens` it's not emitting event	Informational	Resolved
22	Governance can indefinitely disable withdrawal of user funds	Informational	Acknowledged
23	Wrong Function visibility.	Informational	Unresolved
24	Gas Optimization	Informational	Unresolved
25	Gas Optimization	Informational	Unresolved

MEDIUM | RESOLVED

Centralization of emergencyRemoveLiquidity()

In contract FEYTraderJoeProduct, the emergencyRemoveLiquidity() function can be used by a malicious admin to withdraw all the liquidity from the contracts at any point of time or state.

Recommendation:
It is advised to add more decentralization to the contract and roles, such as using a governance mechanism or a multisig.

Comments:
The client assured that they would be using a multisig initially before moving on to a Governance model.

MEDIUM | RESOLVED

Centralization of Pausable and possible Denial of Service

In contract FEYTraderJoeProduct, The global and local pausable functionality can be exploited by a malicious admin to deny users from withdrawing their funds for a very long time. This would also be equivalent to a Denial of Service attack if carried out.

Recommendation:
It is advised to add more decentralization to the contract and roles, such as using a governance mechanism or a multisig.

Comments:
The client assured that they would be using a multisig initially before moving on to a Governance model.

MEDIUM | RESOLVED

Tokens can get stuck while calling deposit()

In contract FEYTraderJoeProduct,
If msg.value != 0 AND address(trancheConfig[_tranche].tokenAddress) != nativeToken, when calling the deposit() and depositFor() functions, it will result in the user's funds being stuck in the contract. It is advised to refund this amount in case the user sends AVAX and the tranchtokenAddress is not nativeToken.

The same issue also exists in the _makeInitialDeposit() function of the factory contract.

Recommendation:

It is advised to add mechanisms in the contract to allow investors from withdrawing their stuck tokens in the contract. Or else refund the tokens immediately in case the tokens are stuck while calling the function.

MEDIUM | RESOLVED

Centralization of setFEYProductImplementation()

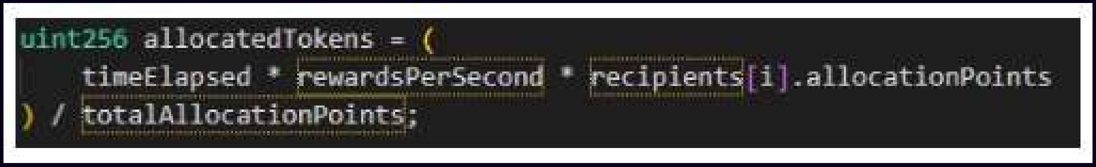
In contract FEYProductFactory, the implementation address of the Product contract can be changed anytime by a malicious admin. This can result in users losing their funds to the attacker via a malicious implementation contract, if new products are deployed using this implementation address.

Recommendation:
It is advised to disallow changing of the implementation contract. It is also advised to add more decentralization to the contract and roles, such as using a governance mechanism or a multisig.

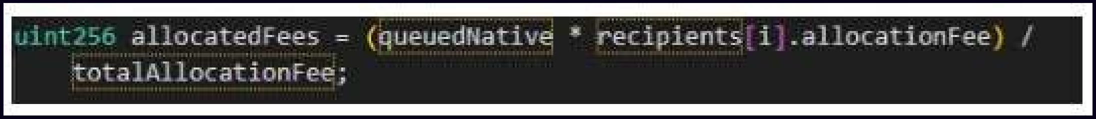
MEDIUM | RESOLVED

Exposure to revert due to wrong operation.

In contract DistributionManager.sol, the function `distributeRewards`



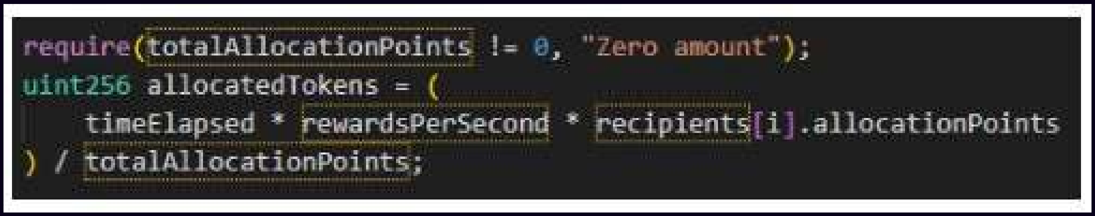
```
uint256 allocatedTokens = (
    timeElapsed * rewardsPerSecond * recipients[i].allocationPoints
) / totalAllocationPoints;
```



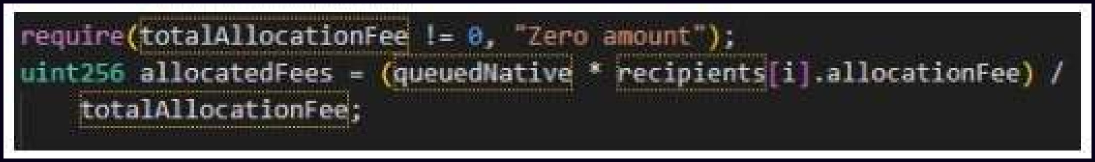
```
uint256 allocatedFees = (queuedNative * recipients[i].allocationFee) /
    totalAllocationFee;
```

is exposed to a panic revert (leads to Denial of service) in some valid cases if the one of `totalAllocationPoints` and `totalAllocationFee` is zero.

Recommendation:
Add zero amount check.



```
require(totalAllocationPoints != 0, "Zero amount");
uint256 allocatedTokens = (
    timeElapsed * rewardsPerSecond * recipients[i].allocationPoints
) / totalAllocationPoints;
```



```
require(totalAllocationFee != 0, "Zero amount");
uint256 allocatedFees = (queuedNative * recipients[i].allocationFee) /
    totalAllocationFee;
```

LOW | RESOLVED

Authorization of only EOAs can break multisigs

The onlyEOAOrRole() modifier in the GACManaged contract, allows only EOAs to interact with the contracts that use this modifier. Smart contracts ideally should NOT be "not allowed" to interact with the client's contracts as it could break the client's smart contract's support with multisig contracts of users, which in turn is not advised for security. This is because usage of multisig contracts or wallets to interact with smart contracts is considered a good security practice.

For example, if a user is using Gnosis safe to interact with the Product contract, the interaction will not be allowed as Gnosis safe(i.e. multisig) is a smart contract.

Recommendation:

It is advised to remove disallowing smart contracts from interacting with the client's smart contracts unless absolutely necessary.

LOW | ACKNOWLEDGED

Bypassing require checks possible

In contract FEYTraderJoeProduct, It is possible to deploy Product contract without the factory contract. If someone accidentally deploys the Product contract without the factory, then all the critical requirement checks used in the _validateProductConfig() can be bypassed.

Recommendation:

It is advised to make the FEYTraderJoeProduct contract abstract in order to avoid this issue.

Comment:

The client stated that If the product contract is deployed by someone (without the factory), then the frontend would need to be exploited to trick the users to deposit funds into the malicious contract. And that Direct interaction with the malicious contract is very unlikely, as the users could easily read the params from the explorer if verified.

LOW | ACKNOWLEDGED

Missing zero address checks

In contract FEYTraderJoeProduct, there is missing zero address for _spToken, _initConfig.configTrancheSr.tokenAddress, _initConfig.configTrancheJr.tokenAddress, _nativeToken, _structPriceOracle, _distributionManager and _yieldSource in the initialize() function.

Recommendation:

It is advised to add zero address checks for the same to avoid incorrect values being assigned.

Comment:

The client stated that the products will be mostly created using the frontend so the users don't have to worry about the addresses. And that even if they added zero address check, there are many other addresses whose private key are unknown. They further added that if users or other protocols are interacting with the contract directly they need to be extremely careful when entering the input params.

LOW | ACKNOWLEDGED

Investors can lose funds due to delay in withdrawal

In contract FEYTraderJoeProduct, the rescueTokens() function allows an admin to withdraw all the tokens from the contract after 3 weeks from the tranche end time. Ideally, the investors should be first allowed to withdraw their funds first, or the contract can push the funds to them before calling rescueTokens(). Otherwise, it could result in a malicious admin withdrawing investor's tokens, in case the investors forget to withdraw their tokens after 3 weeks from the tranche end time.

Recommendation:

It is advised to allow pushing of tokens to the investors or allow automatic withdrawal of tokens (such as with Chainlink keepers) to investors before rescue tokens.

Comment:

The client acknowledged this issue, stating that they'll be using multisig for Governance initially and that the 3 weeks time works as a cooldown period. They said that if the users forget to withdraw their funds, the Struct team will withdraw on their behalf and we will send it to the investor's address.

LOW | ACKNOWLEDGED

Misleading value assignment to poolId

In contract TraderJoeYieldSource, according to the line: 175 in the constructor,

```
poolId = isFarmExists ? _poolId : 0;
```

The poolId to _poolId in case the Farm exists. But it is entirely possible that the poolId is set to 0 even when the Farm exists by passing the _poolId parameter as 0 in the constructor. This would be contradictory to the fact that the poolId is set to 0 when the isFarmExists is false (i.e. the Farm does not exist) according to line: 175.

Recommendation:

It is advised to disallow passing _poolId as 0 in the constructor parameter to avoid unintended issues and logical flaws.

Comment:

The client said that the YieldSource contracts will be deployed manually by the Struct team and that they will be validating the parameters before deployment

LOW | ACKNOWLEDGED

Missing Input Validation in StructPriceOracle

Note: Attacker = (Malicious governance)

Overview: The StructPriceOracle contract is used to fetch the latest price of the given assets using Chainlink's price feed. The contract allows the owner to set or replace sources for the assets. However, the contract does not have proper input validation in the **_setAssetsSources** function, which allows an attacker to add a malicious asset source and control the price returned by the **getAssetPrice** function.

Vulnerability: The vulnerability exists in the **_setAssetsSources** function, where the contract allows the owner to set or replace sources for the assets without proper input validation. An attacker can add a malicious asset source that returns an incorrect price. This can lead to incorrect valuation of the assets, which can cause severe financial loss to the users.

Attack Scenario: An attacker can create a malicious asset source that returns an incorrect price. The attacker can then call the **_setAssetsSources** function with the malicious asset source address and set it as the source for an asset. When the **getAssetPrice** function is called with the asset address, the malicious asset source will return the incorrect price, which can cause the valuation of the assets to be incorrect. This can lead to financial loss for the users.

Impact: The impact of this vulnerability can be severe, as a Malicious Owner can manipulate the price of an asset and cause financial loss to the users. This can also affect the valuation of the assets, which can cause further financial loss. The users can lose trust in the platform, and the reputation of the platform can be damaged.

Recommendation:

To mitigate this vulnerability, the contract should have proper input validation in the **_setAssetsSources** function. The contract should validate that the address of the asset source is a valid Chainlink aggregator address. Additionally, the contract can also implement a whitelist for the asset sources, where the owner can only set the sources from the whitelist. This will prevent the owner from adding a malicious asset source.

Comment:
The client will be using a multisig for governance operations initially. So the scenario is very unlikely to happen as the signers will be some of the industry's trusted parties

LOW | ACKNOWLEDGED

Iterating allProducts array can cause the gas limit to exceed

The variable **allProducts** maintains an array of created products. If an external smart contract attempts to iterate over the array to validate if an address is a valid product or not, the transaction can exceed the gas limit and fail due to the large size of the array.

Recommendation:
User enumerable sets instead of the array. Using enumerable sets provides additional features for validating whether an address is a valid product or not in constant time O(1).
Link: https://docs.openzeppelin.com/contracts/3.x/api/utils#EnumerableSet

LOW | RESOLVED

trancheDurationMax can be less than trancheDurationMin

Description: Although the functions to set the maximum and minimum tranche duration have **onlyRole(GOVERNANCE)** function modifier, decreasing the possibility of having **trancheDurationMax < trancheDurationMin** in the state of the contract, the contract can still have **trancheDurationMax < trancheDurationMin.**

Recommendation:

Add checks in the function **setMinimumTrancheDuration()** and **setMaximumTrancheDuration()** to avoid this state in the contract.

LOW | ACKNOWLEDGED

Fees can be set greater than 100%

The variables **managementFee** and **performanceFee** can be updated using function setManagementFee() and setPerformanceFee() respectively. Although these functions have **onlyRole(GOVERNANCE)** function modifier, the contract allows setting these values to an arbitrarily high value, which could be greater than 100%.

Recommendation:

Add validation while updating fee values in setManagementFee() and setPerformanceFee() such that new values are within acceptable limits, e.g. not more than 100%.

Comment:

Multisig keys will be distributed and the signers will validate the fees before being set

LOW | ACKNOWLEDGED

No check for zero amount

In contract DistributionManager.sol, the function `setRewardsPerSecond`, there is no zero amount check.

Recommendation:

Add the zero amount check

Comment:

It can be zero initially to avoid distributing rewards

INFORMATIONAL | UNRESOLVED

Gas Optimization

In contract DistributionManager.sol, In the function `queueFees`
`` `if` `` check condition is used for zero amount check.
In this case, if the given amount is zero, `queuedNative` isn’t changed, but the transaction will be successful. So it can cause unnecessary gas consumption.

Recommendation:

Add the `require` check condition

INFORMATIONAL | ACKNOWLEDGED

initialize() can be called by anyone

In contract FEYTraderJoeProduct, function initialize() can be called by anyone to initialize the product contract. This can result in incorrect values being used for initialization.

Recommendation:

Although the factory contract immediately initializes the product contract after deployment, it is advised to add appropriate modifiers for the initialize() function as a best practice.

Comment:

The client stated that this would happen when the product contract is deployed separately without the factory contract and stated that their comment on issue 8 would be applicable here.

INFORMATIONAL | ACKNOWLEDGED

Inverted naming of variables

In contract FEYTraderJoeProduct, the naming of leverageThresholdMax and leverageThresholdMin variables is inverted with respect to their execution logic. Inverted naming can lead to confusion during code review.

Recommendation:

It is advised to avoid naming of variables that are the exact opposite of how they behave.

Comment:

The client acknowledged this issue, stating that this is due to the financial terms used in traditional finance, which led them to use the following naming conventions.

INFORMATIONAL | RESOLVED

Critical function `rescueTokens` it's not emmiting event

Critical function rescueTokens it's not emmiting event

It is a good practice that onlyOwner functions always emit event

https://github.com/zokyo-sec/audit-struct-finance-1/blob/audit/zokyo-feb-2023/contracts/protocol/products/traderjoe/FEYTraderJoeProduct.sol#L506

Recommendation:
Add an event to record that owner/governance is rescuing tokens.

INFORMATIONAL | ACKNOWLEDGED

Governance can indefinitely disable withdrawal of user funds

In contract FEYTraderJoeProduct, function withdrawn(), allows a user to withdraw the investment from the product once the tranche is matured. The `gacPausable()` modifier is used in Withdrawn Function to check whether the contract is currently paused or not, if the contract is paused it restrict the deposits of users

Recommendation:
Withdrawals should never be paused because it affects the decentralization nature of the blockchain. Remove gacPausable modifier

Comment:
Governance will be a multisig.

INFORMATIONAL | UNRESOLVED

Wrong Function visibility.

In contract DistributionManager:

- `getRecipients`: this function was not called internally in the contracts, but it was declared as `public`

Recommendation:
declare it as an `external`

- `_validateRecipientConfig`: this function doesn't need to be called externally.

Recommendation:
declare it as an `internal`

In contract GACManaged.sol -

- `pause`: this function was not called internally in the contracts, but it was declared as `public`
- `unpause`: this function was not called internally in the contracts, but it was declared as `public`

Recommendation:
declare it as an `external`

INFORMATIONAL | UNRESOLVED

Gas Optimization

Under the hood of solidity, Booleans (bool) are uint8, which means they use 8 bits of storage. A Boolean can only have two values: True or False. This means that you can store a boolean in only a single bit.

https://github.com/zokyo-sec/audit-struct-finance-1/blob/audit/zokyo-feb-2023/contracts/protocol/common/GACManaged.sol#L30

Recommendation:

Change the uint8 to uint256 to save gas

INFORMATIONAL | UNRESOLVED

Gas Optimization

Change the order of external contract call in _gacPausable function in GACManaged.sol

By swapping the order of the **require** statements, the local pause check will be performed first, which does not require an external contract call. If it fails, the function will revert immediately, saving the gas cost of the external contract call. Only if the local pause check passes, the global pause check will be executed.

https://github.com/zokyo-sec/audit-struct-finance-1/blob/audit/zokyo-feb-2023/contracts/protocol/common/GACManaged.sol#L85

Recommendation:

Change to this :

```
function _gacPausable() private view {
   require(!paused(), Errors.ACE_LOCAL_PAUSED);
   require(!gac.paused(), Errors.ACE_GLOBAL_PAUSED);
}
```

This way, the local pause check will be performed first, and if it fails, the function will revert immediately without incurring the gas cost of the external contract call. If the local pause check passes, only then the global pause check will be executed

	FEYTraderJoeProduct.sol FEYTraderJoeProductFactory.sol TraderJoeYieldSource.sol GACManaged.sol
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

	StructERC1155.sol **StructSPToken.sol** **Constants.sol** **Errors.sol**
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

	Helpers.sol Validation.sol DataTypes.sols
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

We are grateful for the opportunity to work with the Struct Finance team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the Struct Finance team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo