

# SuperLink

## Smart Contract Audit Report Prepared for Coin98



COIN98

---

**Date Issued:** May 27, 2022  
**Project ID:** AUDIT2022032  
**Version:** v1.0  
**Confidentiality Level:** Public

## Report Information

Project ID	AUDIT2022032
Version	v1.0
Client	Coin98
Project	SuperLink
Auditor(s)	Peeraphut Punsuwan Ronnachai Chaipha Sorawish Laovakul
Author(s)	Wachirawit Kanpanluk
Reviewer	Patipon Suwanbol
Confidentiality Level	Public

## Version History

Version	Date	Description	Author(s)
1.0	May 27, 2022	Full report	Wachirawit Kanpanluk

## Contact Information

Company	Inspex
Phone	(+66) 90 888 7186
Telegram	<a href="https://t.me/inspexco">t.me/inspexco</a>
Email	<a href="mailto:audit@inspex.co">audit@inspex.co</a>

## Table of Contents

<b>1. Executive Summary</b>	<b>1</b>
1.1. Audit Result	1
1.2. Disclaimer	1
<b>2. Project Overview</b>	<b>2</b>
2.1. Project Introduction	2
2.2. Scope	3
<b>3. Methodology</b>	<b>4</b>
3.1. Test Categories	4
3.2. Audit Items	5
3.3. Risk Rating	7
<b>4. Summary of Findings</b>	<b>8</b>
<b>5. Detailed Findings Information</b>	<b>10</b>
5.1. Centralized Control of State Variable	10
5.2. Improper Partner Fee Design	12
5.3. Improper msg.value Validation in the swap() Function	17
5.4. Insufficient Logging for Privileged Functions	25
5.5. Reentrancy Attack in swap() Function	27
5.6. Improper Function Visibility	32
<b>6. Appendix</b>	<b>34</b>
6.1. About Inspex	34

## 1. Executive Summary

As requested by Coin98, Inspex team conducted an audit to verify the security posture of the SuperLink smart contracts between May 18, 2022 and May 19, 2022. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of SuperLink 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 low, 2 very low, and 2 info-severity issues. With the project team's prompt response, 1 low, 2 very low and 2 info-severity issues were resolved in the reassessment, while 1 high-severity issue was acknowledged by the team. 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

Coin98 is a Financial Services builder that creates and develops an ecosystem of DeFi protocols, applications, and NFTs on multiple blockchains. The platform can help people to access DeFi services effortlessly.

SuperLink is the platform that provides token swaps service to partners and any user. The partner platform also claims a fee from users when the users swap tokens through the partner platform.

#### Scope Information:

Project Name	SuperLink
Website	<a href="https://coin98.com/">https://coin98.com/</a>
Smart Contract Type	Ethereum Smart Contract
Chain	BNB Smart Chain
Programming Language	Solidity
Category	Dex Aggregator

#### Audit Information:

Audit Method	Whitebox
Audit Date	May 18, 2022 - May 19, 2022
Reassessment Date	May 26, 2022

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 Inspect in detail:

**Initial Audit: (Commit: f3561106c9e2148eb8679460d682a508ed729a2e)**

Contract	Location (URL)
SuperLink	<a href="https://github.com/coin98/coin98-superlink/blob/f3561106c9/contracts/SuperLink.sol">https://github.com/coin98/coin98-superlink/blob/f3561106c9/contracts/SuperLink.sol</a>
SuperLinkPartner	<a href="https://github.com/coin98/coin98-superlink/blob/f3561106c9/contracts/SuperLinkPartner.sol">https://github.com/coin98/coin98-superlink/blob/f3561106c9/contracts/SuperLinkPartner.sol</a>

**Reassessment: (Commit: 329155bace1ddcd4a255ae4ee5d464faabe7e21f)**

Contract	Location (URL)
SuperLink	<a href="https://github.com/coin98/coin98-superlink/blob/329155bace/contracts/SuperLink.sol">https://github.com/coin98/coin98-superlink/blob/329155bace/contracts/SuperLink.sol</a>
SuperLinkPartner	<a href="https://github.com/coin98/coin98-superlink/blob/329155bace/contracts/SuperLinkPartner.sol">https://github.com/coin98/coin98-superlink/blob/329155bace/contracts/SuperLinkPartner.sol</a>

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

### 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](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://inspex.gitbook.io/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](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**.

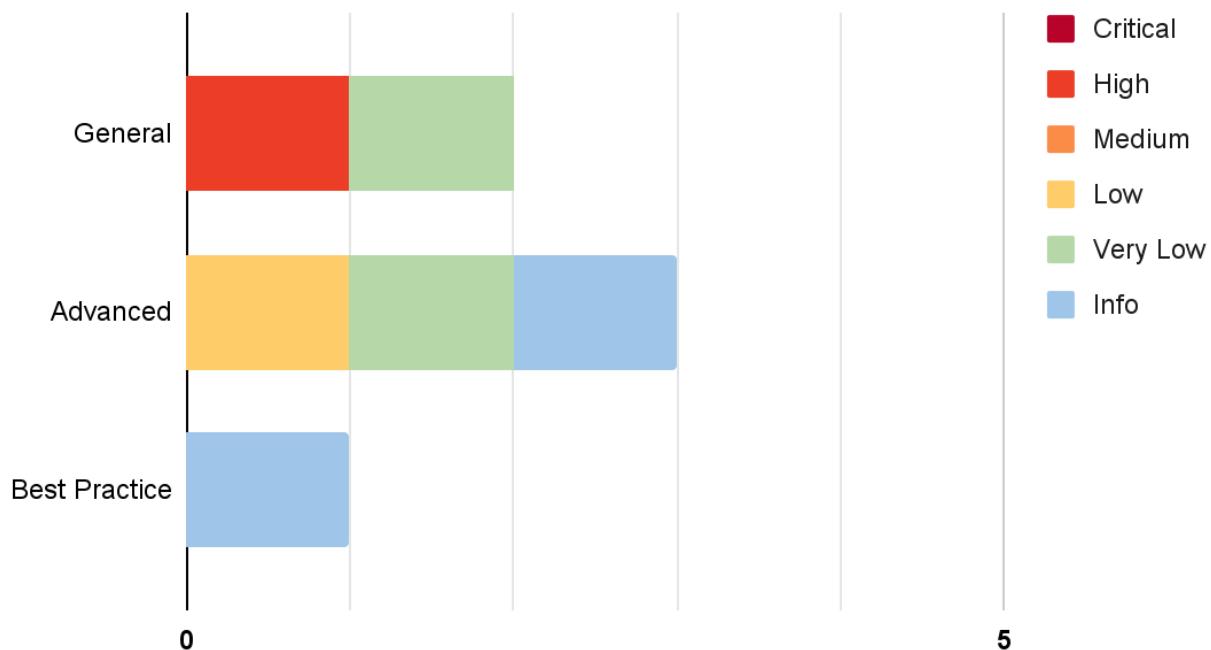
**Severity** is the overall risk of the issue. It can be categorized into five levels: **Very Low**, **Low**, **Medium**, **High**, and **Critical**. It is calculated from the combination of likelihood and impact factors using the matrix below. The severity of findings with no likelihood or impact would 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	Centralized Control of State Variable	General	High	Acknowledged
IDX-002	Improper Partner Fee Design	Advanced	Low	Resolved
IDX-003	Improper msg.value Validation in the swap() Function	Advanced	Very Low	Resolved
IDX-004	Insufficient Logging for Privileged Functions	General	Very Low	Resolved
IDX-005	Reentrancy Attack in swap() Function	Advanced	Info	Resolved
IDX-006	Improper Function Visibility	Best Practice	Info	Resolved

\* The mitigations or clarifications by Coin98 can be found in Chapter 5.

## 5. Detailed Findings Information

### 5.1. Centralized Control of State Variable

ID	IDX-001
Target	SuperLink SuperLinkPartner
Category	General Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	<p><b>Severity: High</b></p> <p><b>Impact: High</b> The controlling authorities can change the critical state variables to gain additional profit. Thus, it is unfair to the other users. For example, the protocol fee in the <b>SuperLink</b> contract can be set to any value by the owner.</p> <p><b>Likelihood: Medium</b> There is nothing to restrict the changes from being done; however, this action can only be done by the privileged roles.</p>
Status	<p><b>Acknowledged</b></p> <p>The Coin98 team has acknowledged and accepted the issue's risk. However, the Coin98 has confirmed that it will notify before any changes have been done.</p>

#### 5.1.1. 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 constraint to prevent the authorities from modifying these variables without notifying the users.

The controllable privileged state update functions are as follows:

Target	Function	Modifier
SuperLink.sol (L:302)	setPartner()	onlyOwner
SuperLink.sol (L:324)	setProtocolFee()	onlyOwner
SuperLink.sol (L:335)	setPartnerFee()	onlyOwner
SuperLinkPartner.sol (L:557)	setPartnerFee()	onlyOwner

### 5.1.2. Remediation

In the ideal case, the critical state variables should not be modifiable to keep the integrity of the smart contract. However, if modifications are needed, Inspect suggests implementing a community-run smart contract governance to control the use of these functions.

If removing the functions or implementing the smart contract governance is not possible, Inspect suggests mitigating the risk of this issue by using a timelock mechanism to delay the changes for a reasonable amount of time, e.g., 24 hours.

## 5.2. Improper Partner Fee Design

ID	IDX-002
Target	SuperLink
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	<p><b>Severity:</b> Low</p> <p><b>Impact:</b> Medium</p> <p>Users may be able to avoid paying the partner fee.</p> <p><b>Likelihood:</b> Low</p> <p>The user is unlikely to acquire the best rate by using the route from the delayed SmartRoute result.</p>
Status	<p><b>Resolved</b></p> <p>The Coin98 team has resolved this issue as suggested by adding the access role with only the partner address that can use the <code>swap()</code> function in commit 329155bace1ddcd4a255ae4ee5d464faabe7e21f.</p>

### 5.2.1. Description

The partner can provide the SmartRoute for users to swap the token at the best rate. The partner also distributes the point and bonus to users who use the `swap()` function with the partner address.

#### SuperLink.sol

```

549 function swap(
550     address _partner,
551     address _tokenIn,
552     address _tokenOut,
553     SwapParameter[] calldata swapList
554 ) public payable {
555     uint256 sizeSwap = swapList.length;
556     uint256 totalOutputAmount = 0;
557     uint256 totalInAmount = 0;
558
559     // Check current balance of SuperLink before swap
560     uint256 currentBalance = IERC20(_tokenOut).balanceOf(address(this));
561
562     for (uint256 i = 0; i < sizeSwap; i++) {
563         SwapParameter calldata swapSelected = swapList[i];
564         uint256 sizePool = swapSelected.pools.length;
565
566         totalOutputAmount = totalOutputAmount.add(

```

```
567         swapSelected.amountOut[sizePool - 1]
568     );
569
570     for (uint256 k = 0; k < sizePool; k++) {
571         bool isLastPool = k == (sizePool - 1);
572         address pool = swapSelected.pools[k];
573
574         if (k == 0) {
575             onTransferFrom(_tokenIn, swapSelected.amountIn, pool);
576             totalInAmount = totalInAmount.add(swapSelected.amountIn);
577         }
578
579         poolSwap(
580             swapSelected.amountOut[k],
581             pool,
582             swapSelected.tokenInOut[k],
583             isLastPool ? address(this) : swapSelected.pools[k + 1]
584         );
585     }
586 }
587
588 uint256 currentBalanceAfter = IERC20(_tokenOut).balanceOf(
589     address(this)
590 );
591
592 // Double check for security check
593 require(
594     currentBalanceAfter.sub(currentBalance) >= totalOutputAmount,
595     "SuperLink: Amount must be matching with total output amount"
596 );
597
598 // Claim system fee for each swap
599 uint256 protocolFee = claimProtocol(totalOutputAmount);
600 uint256 returnAmount = claimPartner(_partner, _tokenOut, protocolFee);
601
602 // Return claim amount to user
603 transferMoney(_tokenOut, returnAmount, msg.sender);
604 emit Swap(
605     msg.sender,
606     _tokenIn,
607     _tokenOut,
608     totalInAmount,
609     returnAmount,
610     protocolFee
611 );
612 }
```

When the users input the partner address parameter, the partner will get the fee from the users.

## SuperLink.sol

```

362 function claimPartner(
363     address _partner,
364     address _tokenOut,
365     uint256 _amount
366 ) internal returns (uint256) {
367     require(
368         _amount > 0,
369         "SuperLink: Amount must be a positive number and greater than zero"
370     );
371
372     if (Partners[_partner].isActive) {
373         bool isSystemFee = _partner == address(this);
374
375         uint256 claimPartnerFee = _amount
376             .mul(
377                 isSystemFee
378                     ? PARTNER_FEE
379                     : PartnerSuperLink(_partner).PARTNER_FEE()
380             )
381             .div(Percent);
382
383         // Transfer fee to partner
384         if (!isSystemFee) {
385             transferMoney(_tokenOut, claimPartnerFee, _partner);
386         }
387         emit ClaimPartnerFee(_partner, _tokenOut, claimPartnerFee);
388         return _amount.sub(claimPartnerFee);
389     }
390     return _amount;
391 }
```

However, the `swap()` function allows any user to swap by inputting the `swapList` parameter by themselves and inputting the `_partner` parameter with `address(0)` to bypass the fee in the `swap()` function. It's resulting in the partner not getting the fee.

### 5.2.2. Remediation

For the benefit of the partner who provides the smart route, Inspex suggests preventing users from bypassing partner fees by adding the access role with only the partner address that can use the `swap()` function.

## SuperLink.sol

```

549 function swap(
550     address _partner,
551     address _tokenIn,
552     address _tokenOut,
```

```
553     SwapParameter[] calldata swapList
554 ) public payable {
555
556     require(Partners[_partner].isActive, "SuperLink: Only Partner is
557 required");
558
559     uint256 sizeSwap = swapList.length;
560     uint256 totalOutputAmount = 0;
561     uint256 totalInAmount = 0;
562
563     // Check current balance of SuperLink before swap
564     uint256 currentBalance = IERC20(_tokenOut).balanceOf(address(this));
565
566     for (uint256 i = 0; i < sizeSwap; i++) {
567         SwapParameter calldata swapSelected = swapList[i];
568         uint256 sizePool = swapSelected.pools.length;
569
570         totalOutputAmount = totalOutputAmount.add(
571             swapSelected.amountOut[sizePool - 1]
572         );
573
574         for (uint256 k = 0; k < sizePool; k++) {
575             bool isLastPool = k == (sizePool - 1);
576             address pool = swapSelected.pools[k];
577
578             if (k == 0) {
579                 onTransferFrom(_tokenIn, swapSelected.amountIn, pool);
580                 totalInAmount = totalInAmount.add(swapSelected.amountIn);
581             }
582
583             poolSwap(
584                 swapSelected.amountOut[k],
585                 pool,
586                 swapSelected.tokenInOut[k],
587                 isLastPool ? address(this) : swapSelected.pools[k + 1]
588             );
589         }
590
591         uint256 currentBalanceAfter = IERC20(_tokenOut).balanceOf(
592             address(this)
593         );
594
595         // Double check for security check
596         require(
597             currentBalanceAfter.sub(currentBalance) >= totalOutputAmount,
598             "SuperLink: Amount must be matching with total output amount"
```

```
599 );
600
601 // Claim system fee for each swap
602 uint256 protocolFee = claimProtocol(totalOutputAmount);
603 uint256 returnAmount = claimPartner(_partner, _tokenOut, protocolFee);
604
605 // Return claim amount to user
606 transferMoney(_tokenOut, returnAmount, msg.sender);
607 emit Swap(
608     msg.sender,
609     _tokenIn,
610     _tokenOut,
611     totalInAmount,
612     returnAmount,
613     protocolFee
614 );
615 }
```

## 5.3. Improper msg.value Validation in the swap() Function

ID	IDX-003
Target	SuperLink
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	<p><b>Severity:</b> Very Low</p> <p><b>Impact:</b> Low</p> <p>The attacker can drain the native token from the contract by using the <code>swap()</code> function.</p> <p><b>Likelihood:</b> Low</p> <p>It is not likely that there are native tokens in the contract.</p>
Status	<p><b>Resolved</b></p> <p>The Coin98 team has resolved this issue as suggested by adding the validation mechanism to ensure that the total sum of the <code>amountIn</code> value is equal to the <code>callvalue()</code> value in commit 329155bace1ddcd4a255ae4ee5d464faabe7e21f.</p>

### 5.3.1. Description

The `swap()` function allows users to swap different tokens with the ability to specify pool lists to swap, and users can also swap it multiple times at once by passing the amount of swap and token with the `swapList` parameter.

When users enter the `_tokenIn` parameter as a WBNB token to swap with a native token, the `SuperLink` contract will call the `onTransferFrom()` function and wrap the native token to the WBNB token, then transfer to the first pool, at the following lines 575 and 579 respectively.

#### SuperLink.sol

```

549 function swap(
550     address _partner,
551     address _tokenIn,
552     address _tokenOut,
553     SwapParameter[] calldata swapList
554 ) public payable {
555     uint256 sizeSwap = swapList.length;
556     uint256 totalOutputAmount = 0;
557     uint256 totalInAmount = 0;
558
559     // Check current balance of SuperLink before swap
560     uint256 currentBalance = IERC20(_tokenOut).balanceOf(address(this));
561
562     for (uint256 i = 0; i < sizeSwap; i++) {

```

```
563     SwapParameter calldata swapSelected = swapList[i];
564     uint256 sizePool = swapSelected.pools.length;
565
566     totalOutputAmount = totalOutputAmount.add(
567         swapSelected.amountOut[sizePool - 1]
568     );
569
570     for (uint256 k = 0; k < sizePool; k++) {
571         bool isLastPool = k == (sizePool - 1);
572         address pool = swapSelected.pools[k];
573
574         if (k == 0) {
575             onTransferFrom(_tokenIn, swapSelected.amountIn, pool);
576             totalInAmount = totalInAmount.add(swapSelected.amountIn);
577         }
578
579         poolSwap(
580             swapSelected.amountOut[k],
581             pool,
582             swapSelected.tokenInOut[k],
583             isLastPool ? address(this) : swapSelected.pools[k + 1]
584         );
585     }
586 }
587
588 uint256 currentBalanceAfter = IERC20(_tokenOut).balanceOf(
589     address(this)
590 );
591
592 // Double check for security check
593 require(
594     currentBalanceAfter.sub(currentBalance) >= totalOutputAmount,
595     "SuperLink: Amount must be matching with total output amount"
596 );
597
598 // Claim system fee for each swap
599 uint256 protocolFee = claimProtocol(totalOutputAmount);
600 uint256 returnAmount = claimPartner(_partner, _tokenOut, protocolFee);
601
602 // Return claim amount to user
603 transferMoney(_tokenOut, returnAmount, msg.sender);
604 emit Swap(
605     msg.sender,
606     _tokenIn,
607     _tokenOut,
608     totalInAmount,
609     returnAmount,
```

```
610     protocolFee  
611     );  
612 }
```

Before transferring, the `onTransferFrom()` function will wrap the native token to the WBWNB token by calling the `deposit()` function in the WBBNB contract at line 491.

## SuperLink.sol

```

474         mstore(emptyPtr, _TRANSFER_FROM_CALL_SELECTOR_32)
475         mstore(add(emptyPtr, 0x4), caller())
476         mstore(add(emptyPtr, 0x24), _receiver)
477         mstore(add(emptyPtr, 0x44), _amount)
478         if iszero(call(gas(), _token, 0, emptyPtr, 0x64, 0, 0)) {
479             reRevert()
480         }
481     }
482     default {
483         if iszero(eq(_amount, callvalue())) {
484             revertWithReason(
485                 0x00000011696e76616c6964206d73672e76616c75650000000000000000000000000000,
486                 0x55
487             ) // "Invalid sender msg.value"
488         }
489
490         mstore(emptyPtr, _WETH_DEPOSIT_CALL_SELECTOR_32)
491         if iszero(call(gas(), _WETH, _amount, emptyPtr, 0x4, 0, 0)) {
492             reRevert()
493         }
494
495         mstore(emptyPtr, _ERC20_TRANSFER_CALL_SELECTOR_32)
496         mstore(add(emptyPtr, 0x4), _receiver)
497         mstore(add(emptyPtr, 0x24), _amount)
498         if iszero(call(gas(), _WETH, 0, emptyPtr, 0x44, 0, 0)) {
499             reRevert()
500         }
501     }
502 }
503 }
```

The problem is when users are doing multiple swaps at once by calling the `swap()` function and passing a list in the `swapList` parameter, the `onTransferFrom()` function will check that the `amountIn` value is equal to the `callvalue()` value, which will cause an unexpected result.

For example, if users input the `swapList` parameter for a swap with WBWNB token two times with the `amountIn` parameter equal to 1, just 1 BNB is sent in. The `iszero(eq( amount, callvalue())` condition will be false since the `amountIn` is always equal to the `callvalue()`.

As a result, the contract's BNB will be swapped for the `_tokenOut` tokens, which will be transferred to the user.

However, the fee will be collected as a WBWNB token instead of BNB, so there may be no BNB left in the SuperLink contract.

### 5.3.2. Remediation

InspeX suggests adding the validation mechanism to ensure that the total sum of the `amountIn` value is equal to the `callvalue()` value, for example:

- Remove the check with the `callvalue()` condition in the `onTransferFrom()` function at lines 483 - 488.

## SuperLink.sol

```

473
474     mstore(emptyPtr, _TRANSFER_FROM_CALL_SELECTOR_32)
475     mstore(add(emptyPtr, 0x4), caller())
476     mstore(add(emptyPtr, 0x24), _receiver)
477     mstore(add(emptyPtr, 0x44), _amount)
478     if iszero(call(gas(), _token, 0, emptyPtr, 0x64, 0, 0)) {
479         reRevert()
480     }
481 }
482 default {
483     mstore(emptyPtr, _WETH_DEPOSIT_CALL_SELECTOR_32)
484     if iszero(call(gas(), _WETH, _amount, emptyPtr, 0x4, 0, 0)) {
485         reRevert()
486     }
487
488     mstore(emptyPtr, _ERC20_TRANSFER_CALL_SELECTOR_32)
489     mstore(add(emptyPtr, 0x4), _receiver)
490     mstore(add(emptyPtr, 0x24), _amount)
491     if iszero(call(gas(), _WETH, 0, emptyPtr, 0x44, 0, 0)) {
492         reRevert()
493     }
494 }
495 }
496 }
```

- Ensure that the transferred BNB equals the sum of the `amountIn` value in lines 588 - 593.

### SuperLink.sol

```

549 function swap(
550     address _partner,
551     address _tokenIn,
552     address _tokenOut,
553     SwapParameter[] calldata swapList
554 ) public payable {
555     uint256 sizeSwap = swapList.length;
556     uint256 totalOutputAmount = 0;
557     uint256 totalInAmount = 0;
558
559     // Check current balance of SuperLink before swap
560     uint256 currentBalance = IERC20(_tokenOut).balanceOf(address(this));
561
562     for (uint256 i = 0; i < sizeSwap; i++) {
563         SwapParameter calldata swapSelected = swapList[i];
564         uint256 sizePool = swapSelected.pools.length;
565
566         totalOutputAmount = totalOutputAmount.add(
567             swapSelected.amountOut[sizePool - 1]
```

```
568 );
569
570     for (uint256 k = 0; k < sizePool; k++) {
571         bool isLastPool = k == (sizePool - 1);
572         address pool = swapSelected.pools[k];
573
574         if (k == 0) {
575             onTransferFrom(_tokenIn, swapSelected.amountIn, pool);
576             totalInAmount = totalInAmount.add(swapSelected.amountIn);
577         }
578
579         poolSwap(
580             swapSelected.amountOut[k],
581             pool,
582             swapSelected.tokenInOut[k],
583             isLastPool ? address(this) : swapSelected.pools[k + 1]
584         );
585     }
586 }
587
588 if (_tokenIn == address(uint160(_WETH))){
589     require(
590         msg.value >= totalInAmount,
591         "SuperLink: Amount must be matching with total input amount"
592     );
593 }
594
595 uint256 currentBalanceAfter = IERC20(_tokenOut).balanceOf(
596     address(this)
597 );
598
599 // Double check for security check
600 require(
601     currentBalanceAfter.sub(currentBalance) >= totalOutputAmount,
602     "SuperLink: Amount must be matching with total output amount"
603 );
604
605 // Claim system fee for each swap
606 uint256 protocolFee = claimProtocol(totalOutputAmount);
607 uint256 returnAmount = claimPartner(_partner, _tokenOut, protocolFee);
608
609 // Return claim amount to user
610 transferMoney(_tokenOut, returnAmount, msg.sender);
611 emit Swap(
612     msg.sender,
613     _tokenIn,
614     _tokenOut,
```

```
615     totalInAmount,  
616     returnAmount,  
617     protocolFee  
618 );  
619 }
```

## 5.4. Insufficient Logging for Privileged Functions

ID	IDX-004
Target	SuperLink SuperLinkPartner
Category	General Smart Contract Vulnerability
CWE	CWE-778: Insufficient Logging
Risk	<p><b>Severity:</b> Very Low</p> <p><b>Impact:</b> Low</p> <p>Privileged functions' executions cannot be monitored easily by the users.</p> <p><b>Likelihood:</b> Low</p> <p>It is not likely that the execution of the privileged functions will be a malicious action.</p>
Status	<p><b>Resolved</b></p> <p>The Coin98 team has resolved this issue as suggested by emitting events for the execution of privileged functions in commit 329155bace1ddcd4a255ae4ee5d464faabe7e21f.</p>

### 5.4.1. 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 could withdraw token by executing the `withdrawMultiple()` function in the `SuperLink` contract, and no events are emitted, for example:

#### SuperLink.sol

```

616 function withdrawMultiple(address[] calldata tokens) public onlyOwner {
617     for (uint256 i = 0; i < tokens.length; i++) {
618         if (tokens[i] == address(0)) {
619             payable(msg.sender).transfer(address(this).balance);
620         } else {
621             IERC20 token = IERC20(tokens[i]);
622
623             uint256 tokenBalance = token.balanceOf(address(this));
624             if (tokenBalance > 0) {
625                 token.safeTransfer(msg.sender, tokenBalance);
626             }
627         }
628     }
629 }
```

The privileged functions without sufficient logging are as follows:

File	Contract	Function
SuperLink.sol (L:616)	SuperLink	withdrawMultiple()
SuperLinkPartner.sol (L:588)	SuperLinkPartner	withdrawMultiple()

### 5.4.2. Remediation

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

#### SuperLink.sol

```

616 event WithdrawToken(address[] calldata tokens);
617 function withdrawMultiple(address[] calldata tokens) public onlyOwner {
618     for (uint256 i = 0; i < tokens.length; i++) {
619         if (tokens[i] == address(0)) {
620             payable(msg.sender).transfer(address(this).balance);
621         } else {
622             IERC20 token = IERC20(tokens[i]);
623
624             uint256 tokenBalance = token.balanceOf(address(this));
625             if (tokenBalance > 0) {
626                 token.safeTransfer(msg.sender, tokenBalance);
627             }
628         }
629     }
630     emit WithdrawToken(tokens);
631 }
```

## 5.5. Reentrancy Attack in swap() Function

ID	IDX-005
Target	SuperLink
Category	Advanced Smart Contract Vulnerability
CWE	CWE-841: Improper Enforcement of Behavioral Workflow
Risk	<b>Severity:</b> Info <b>Impact:</b> None <b>Likelihood:</b> None
Status	<b>Resolved</b> The Coin98 team has resolved this issue as suggested by using OpenZeppelin's <code>ReentrancyGuard</code> contract and adding the <code>nonReentrant</code> modifier to the <code>swap()</code> functions in commit <a href="#">329155bace1ddcd4a255ae4ee5d464faabe7e21f</a> .

### 5.5.1. Description

The `swap()` function in the `SuperLink` contract allows users to manually swap tokens by specifying the pool address and passing it to the `poolSwap()` function, resulting in users inputting the malicious pool address that calls the `swap()` function again, causing a reentrancy attack.

#### SuperLink.sol

```

549 function swap(
550     address _partner,
551     address _tokenIn,
552     address _tokenOut,
553     SwapParameter[] calldata swapList
554 ) public payable {
555     uint256 sizeSwap = swapList.length;
556     uint256 totalOutputAmount = 0;
557     uint256 totalInAmount = 0;
558
559     // Check current balance of SuperLink before swap
560     uint256 currentBalance = IERC20(_tokenOut).balanceOf(address(this));
561
562     for (uint256 i = 0; i < sizeSwap; i++) {
563         SwapParameter calldata swapSelected = swapList[i];
564         uint256 sizePool = swapSelected.pools.length;
565
566         totalOutputAmount = totalOutputAmount.add(
567             swapSelected.amountOut[sizePool - 1]
568         );

```

```
569
570     for (uint256 k = 0; k < sizePool; k++) {
571         bool isLastPool = k == (sizePool - 1);
572         address pool = swapSelected.pools[k];
573
574         if (k == 0) {
575             onTransferFrom(_tokenIn, swapSelected.amountIn, pool);
576             totalInAmount = totalInAmount.add(swapSelected.amountIn);
577         }
578
579         poolSwap(
580             swapSelected.amountOut[k],
581             pool,
582             swapSelected.tokenInOut[k],
583             isLastPool ? address(this) : swapSelected.pools[k + 1]
584         );
585     }
586 }
587
588 uint256 currentBalanceAfter = IERC20(_tokenOut).balanceOf(
589     address(this)
590 );
591
592 // Double check for security check
593 require(
594     currentBalanceAfter.sub(currentBalance) >= totalOutputAmount,
595     "SuperLink: Amount must be matching with total output amount"
596 );
597
598 // Claim system fee for each swap
599 uint256 protocolFee = claimProtocol(totalOutputAmount);
600 uint256 returnAmount = claimPartner(_partner, _tokenOut, protocolFee);
601
602 // Return claim amount to user
603 transferMoney(_tokenOut, returnAmount, msg.sender);
604 emit Swap(
605     msg.sender,
606     _tokenIn,
607     _tokenOut,
608     totalInAmount,
609     returnAmount,
610     protocolFee
611 );
612 }
```

## SuperLink.sol

```

510 function poolSwap(
511     uint256 _amount,
512     address _pool,
513     bool _tokenInOut,
514     address _receiver
515 ) internal {
516     assembly {
517         let emptyPtr := mload(0x40)
518         mstore(0x40, add(emptyPtr, 0xc0))
519
520         function reRevert() {
521             returndatacopy(0, 0, returndatasize())
522             revert(0, returndatasize())
523         }
524         mstore(emptyPtr, _UNISWAP_PAIR_SWAP_CALL_SELECTOR_32)
525
526         switch iszero(_tokenInOut)
527         case 0 {
528             mstore(add(emptyPtr, 0x04), 0)
529             mstore(add(emptyPtr, 0x24), _amount)
530         }
531         default {
532             mstore(add(emptyPtr, 0x04), _amount)
533             mstore(add(emptyPtr, 0x24), 0)
534         }
535         mstore(add(emptyPtr, 0x44), _receiver)
536         mstore(add(emptyPtr, 0x64), 0x80)
537         mstore(add(emptyPtr, 0x84), 0)
538         if iszero(call(gas(), _pool, 0, emptyPtr, 0xa4, 0, 0)) {
539             reRevert()
540         }
541     }
542 }
```

Before the reentrancy attack is performed, the **SuperLink** contract will store the **tokenOut** token balance in the **currentBalance** variable.

During the reentrancy attack, the malicious pool will be re-calling the **swap()** function again to swap some tokens; Therefore, the swap fee will be stored in the **SuperLink** contract.

After the malicious pool process is done, the **SuperLink** contract's **tokenOut** token balance will be increased, which causes the **SuperLink** contract to incorrectly assume that the swap fee produced by the reentrancy attack is due to swaps and will be transferred out of the pool.

As a result, if the **totalAmountOut** matches the outcome of **currentBalance - currentBalanceAfter**, the **SuperLink** contract will compute the **totalAmountOut** fee before transferring the remaining amount to

`msg.sender`; this means users will pay the platform fee less than usual (the partner fee will still be paid normally).

However, the attacker will have no benefit from the platform. But, if the partner distributes rewards to users who use the partner platform to swap tokens, the attacker can use the reentrancy attack to obtain points or rewards from the partner without having to pay the platform fee.

### 5.5.2. Remediation

Inspex suggests using OpenZeppelin's `ReentrancyGuard` contract and adding the `nonReentrant` modifier to the `swap()` functions , for example:

#### SuperLink.sol

```

549 function swap(
550     address _partner,
551     address _tokenIn,
552     address _tokenOut,
553     SwapParameter[] calldata swapList
554 ) public nonReentrant payable {
555     uint256 sizeSwap = swapList.length;
556     uint256 totalOutputAmount = 0;
557     uint256 totalInAmount = 0;
558
559     // Check current balance of SuperLink before swap
560     uint256 currentBalance = IERC20(_tokenOut).balanceOf(address(this));
561
562     for (uint256 i = 0; i < sizeSwap; i++) {
563         SwapParameter calldata swapSelected = swapList[i];
564         uint256 sizePool = swapSelected.pools.length;
565
566         totalOutputAmount = totalOutputAmount.add(
567             swapSelected.amountOut[sizePool - 1]
568         );
569
570         for (uint256 k = 0; k < sizePool; k++) {
571             bool isLastPool = k == (sizePool - 1);
572             address pool = swapSelected.pools[k];
573
574             if (k == 0) {
575                 onTransferFrom(_tokenIn, swapSelected.amountIn, pool);
576                 totalInAmount = totalInAmount.add(swapSelected.amountIn);
577             }
578
579             poolSwap(
580                 swapSelected.amountOut[k],
581                 pool,
582                 swapSelected.tokenInOut[k],

```

```
583         isLastPool ? address(this) : swapSelected.pools[k + 1]
584     );
585 }
586 }
587
588 uint256 currentBalanceAfter = IERC20(_tokenOut).balanceOf(
589     address(this)
590 );
591
592 // Double check for security check
593 require(
594     currentBalanceAfter.sub(currentBalance) >= totalOutputAmount,
595     "SuperLink: Amount must be matching with total output amount"
596 );
597
598 // Claim system fee for each swap
599 uint256 protocolFee = claimProtocol(totalOutputAmount);
600 uint256 returnAmount = claimPartner(_partner, _tokenOut, protocolFee);
601
602 // Return claim amount to user
603 transferMoney(_tokenOut, returnAmount, msg.sender);
604 emit Swap(
605     msg.sender,
606     _tokenIn,
607     _tokenOut,
608     totalInAmount,
609     returnAmount,
610     protocolFee
611 );
612 }
```

## 5.6. Improper Function Visibility

ID	IDX-006
Target	SuperLink SuperLinkPartner
Category	Smart Contract Best Practice
CWE	CWE-710: Improper Adherence to Coding Standards
Risk	<b>Severity:</b> Info <b>Impact:</b> None <b>Likelihood:</b> None
Status	<b>Resolved</b> The Coin98 team has resolved this issue as suggested by changing all functions' visibility to <code>external</code> if they are not called from any functions in the contract at commit <code>329155bace1ddcd4a255ae4ee5d464faabe7e21f</code> .

### 5.6.1. Description

Public functions that are never called internally by the contract itself should have external visibility. This improves the readability of the contract, allowing clear distinction between functions that are externally used and functions that are also called internally.

The following source code shows that the `setProtocolFee()` function on the `SuperLink` contact is set to public and it is never called from any internal function.

#### SuperLink.sol

```

324 function setProtocolFee(uint256 _protocolFee) public onlyOwner {
325     require(
326         _protocolFee > 0,
327         "SuperLink: Fee must be a positive number and greater than zero"
328     );
329     PROTOCOL_FEE = _protocolFee;
330     emit SetProtocolFee(_protocolFee);
331 }
```

The following table contains all functions that have public visibility and never be called from any internal function.

File	Contract	Function
SuperLink.sol (L:324)	SuperLink	setProtocolFee()
SuperLink.sol (L:335)	SuperLink	setPartnerFee()
SuperLink.sol (L:549)	SuperLink	swap()
SuperLink.sol (L:616)	SuperLink	withdrawMultiple()
SuperLinkPartner.sol (L:577)	SuperLinkPartner	setPartnerFee()
SuperLinkPartner.sol (L:588)	SuperLinkPartner	withdrawMultiple()

## 5.6.2. Remediation

Inspex suggests changing all functions' visibility to external if they are not called from any internal function, as shown in the following example:

### SuperLink.sol

```

324 function setProtocolFee(uint256 _protocolFee) external onlyOwner {
325     require(
326         _protocolFee > 0,
327         "SuperLink: Fee must be a positive number and greater than zero"
328     );
329     PROTOCOL_FEE = _protocolFee;
330     emit SetProtocolFee(_protocolFee);
331 }
```

## 6. Appendix

### 6.1. About Inspect



## CYBERSECURITY PROFESSIONAL SERVICE

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	<a href="https://inspex.co">https://inspex.co</a>
Twitter	<a href="https://twitter.com/InspexCo">@InspexCo</a>
Facebook	<a href="https://www.facebook.com/InspexCo">https://www.facebook.com/InspexCo</a>
Telegram	<a href="https://t.me/inspex_announcement">@inspex_announcement</a>



**inspex**

CYBERSECURITY PROFESSIONAL SERVICE