Changer

Smart Contract Security Audit

V1.1

No. 202303131126

Mar 13th, 2023

Contents

Summary of Audit Results **1**

1 Overview **3**

1.1 Project Overview 3

dex-contracts/actions/BaseActions.sol 3

dex-contracts/actions/LowLevelCallActions.sol 3

dex-contracts/actions/lib/*.sol 3

1.2 Audit Overview 3

2 Findings **4**

[Changer-1] *lowLevelCall* function design risk 5

[Changer-2] _*checkAllowance* function is improperly designed 6

3 Appendix **7**

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts 7

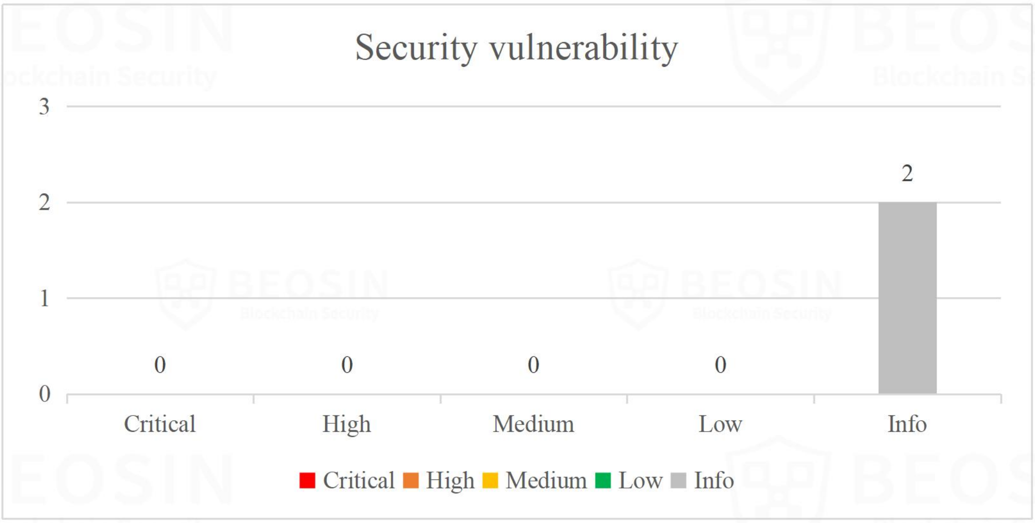
3.2 Audit Categories 9

3.3 Disclaimer 11

3.4 About Beosin 12

Summary of Audit Results

After auditing, 2 info items were identified in the Changer project. Specific audit details will be presented in the **Findings** section. Users should pay attention to the following aspects when interacting with this project:



*Notes:

● **Risk Description:**

In this project, there are three points that users need to pay attention to:

1. The offline business of swapWithCompensation in the project is not within the scope of the audit, and the deployment in the contract adopts the agent upgrade mode, and the logic implementation contract after the upgrade is not within the scope of the audit.

2. The second point is that after the Handler contract (including the historical contract) is wrongly approved, there is a risk of being constructed as a transferFrom. However, users usually do not approve the Handler contract, and it is recommended that users do not approve the Router contract too much.

3. The third point is that the *swap* function in the Router contract depends on the splicing of callData. If the user makes a mistake during use, the funds may remain in the Router contract, and any other account can transfer the wrong funds.

- **Project Description:**

1. **Business overview**

Changer is an automatic aggregator of multiple trading (DEX) platforms. Users can use the *swap* function to select the dex trading platform that best suits their needs for exchange, including Balancer, Bancor, Curve Swap, DoDo, Uniswap, SushiSwap, and ChangerPro. The Changer project adopts the agent upgrade mode deployment model, and the logic realizes that the contract can be upgraded. The contract owner of the Changer has a suspend function that can be used to temporarily stop the token exchange function in case of an emergency.

1 Overview

1.1 Project Overview

Project Name	Changer
Platform	Ethereum&BSC&Polygon
Audit link	https://github.com/changerio/aggregator-contracts/tree/feat/multi-chain#audit-target-contracts
Audit scope	dex-contracts/actions/BaseActions.sol dex-contracts/actions/LowLevelCallActions.sol dex-contracts/actions/lib/*.sol aggregator-contracts/interfaces/*.sol aggregator-contracts/lib/*.sol aggregator-contracts/actions/common/*.sol aggregator-contracts/handler/*.sol aggregator-contracts/router/ChangerSwapRouterV3.sol

Commit Hash	fc719130beab88ee2a078964e3d968ea7822676c(initial)
	672122286aef7852a1fdb80d3f729bb15411e11b(final)

1.2 Audit Overview

Audit work duration: Mar 1, 2023 – Mar 13, 2023

Audit methods: Formal Verification, Static Analysis, Typical Case Testing and Manual Review.

Audit team: Beosin Security Team.

2 Findings

Index	Risk description	Severity level	Status
Changer-1	*lowLevelCall* function design risk	Info	Partially Fixed
Changer-2	_*checkAllowance* function is improperly designed	Info	Acknowledged

Status Notes:

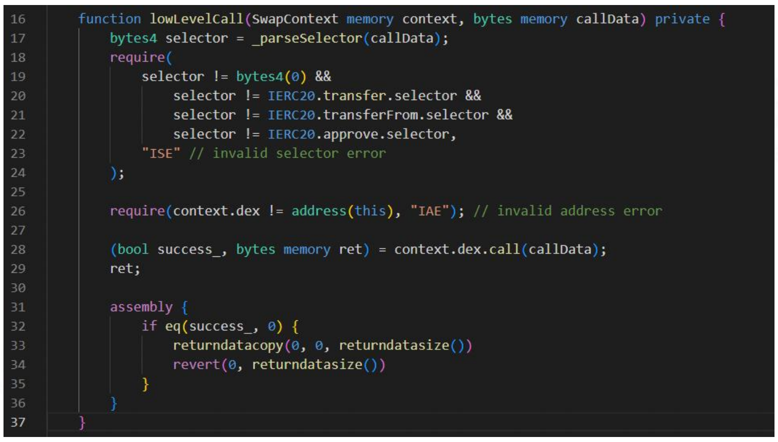
1. Changer-1 is partially fixed and common Token functions are disabled in the blacklist, but it is still recommended that the project party pay attention to use.

2. Changer-2 is nor fixed and the approve value may be too large.

Finding Details:

[Changer-1] *lowLevelCall* function design risk

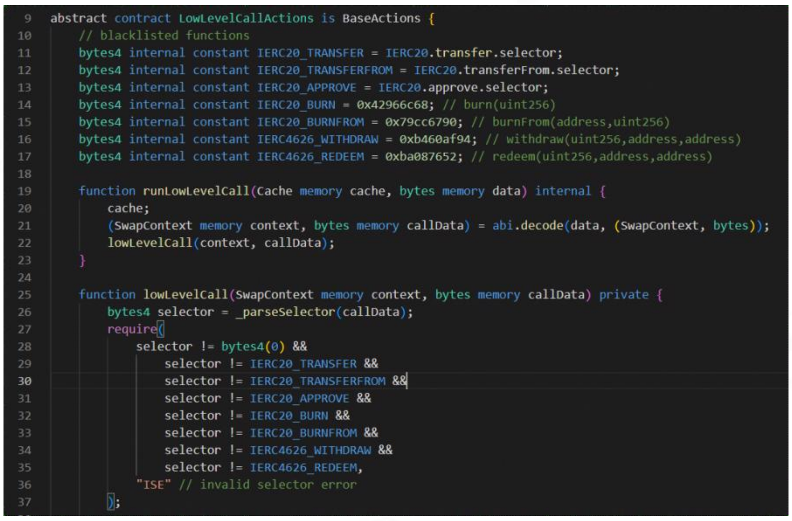
Severity Level	Info
Type	Business Security
Lines	LowLevelCallActions.sol#L17-24
Description	In the *lowLevelCall* function in the LowLevelCallActions contract, a blacklist is used to restrict function calls. From the current contract design, the handler contract should not involve risks. In terms of function points here, there may be risks in blacklist restrictions, such as calling the *burnFrom* function, as long as the user approve the handler contract, the attacker can call the logic to delete the user's tokens.



```
    function lowLevelCall(SwapContext memory context, bytes memory callData) private {
        bytes4 selector = _parseSelector(callData);
        require(
            selector != bytes4(0) &&
                selector != IERC20.transfer.selector &&
                selector != IERC20.transferFrom.selector &&
                selector != IERC20.approve.selector,
            "ISE" // invalid selector error
        );

        require(context.dex != address(this), "IAE"); // invalid address error

        (bool success_, bytes memory ret) = context.dex.call(callData);
        ret;

        assembly {
            if eq(success_, 0) {
                returndatacopy(0, 0, returndatasize())
                revert(0, returndatasize())
            }
        }
    }
```

Figure 1 Source code of *lowLevelCall* function(unfixed)

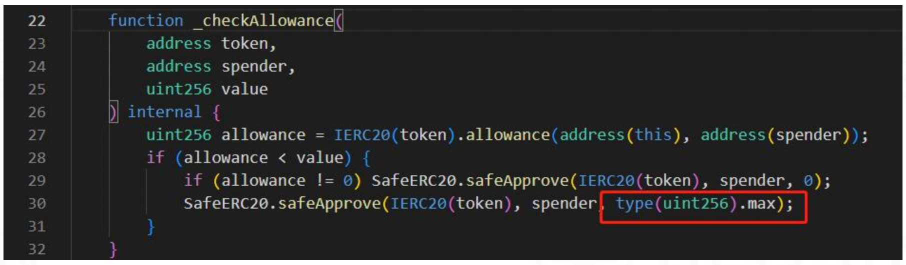
Recommendations	It is not recommended to use the *lowLevelCall* function.
Status	Partially Fixed.



```
abstract contract LowLevelCallActions is BaseActions {
    // blacklisted functions
    bytes4 internal constant IERC20_TRANSFER = IERC20.transfer.selector;
    bytes4 internal constant IERC20_TRANSFERFROM = IERC20.transferFrom.selector;
    bytes4 internal constant IERC20_APPROVE = IERC20.approve.selector;
    bytes4 internal constant IERC20_BURN = 0x42966c68; // burn(uint256)
    bytes4 internal constant IERC20_BURNFROM = 0x79cc6790; // burnFrom(address,uint256)
    bytes4 internal constant IERC4626_WITHDRAW = 0xb460af94; // withdraw(uint256,address,address)
    bytes4 internal constant IERC4626_REDEEM = 0xba087652; // redeem(uint256,address,address)

    function runLowLevelCall(Cache memory cache, bytes memory data) internal {
        cache;
        (SwapContext memory context, bytes memory callData) = abi.decode(data, (SwapContext, bytes));
        lowLevelCall(context, callData);
    }

    function lowLevelCall(SwapContext memory context, bytes memory callData) private {
        bytes4 selector = _parseSelector(callData);
        require(
            selector != bytes4(0) &&
                selector != IERC20_TRANSFER &&
                selector != IERC20_TRANSFERFROM &&
                selector != IERC20_APPROVE &&
                selector != IERC20_BURN &&
                selector != IERC20_BURNFROM &&
                selector != IERC4626_WITHDRAW &&
                selector != IERC4626_REDEEM,
            "ISE" // invalid selector error
        );
```

Figure 2 Source code of *lowLevelCall* function(partially fixed)

[Changer-2] _*checkAllowance* function is improperly designed

Severity Level	Info
Type	Business Security
Lines	BaesActions.sol#L22-32
Description	In the _*checkAllowance* function in the BaseActions contract, When the approve value is not enough, the spender address will be set to zero, and then the maximum value will be assigned. From a security point of view, there is no need to set it to max, just set the approve value to value.  Figure 3 Source code of _*checkAllowance* function
Recommendations	It is not recommended set the approve value to value instead of max.
Status	Acknowledged.

```
function _checkAllowance(
    address token,
    address spender,
    uint256 value
) internal {
    uint256 allowance = IERC20(token).allowance(address(this), address(spender));
    if (allowance < value) {
        if (allowance != 0) SafeERC20.safeApprove(IERC20(token), spender, 0);
        SafeERC20.safeApprove(IERC20(token), spender, type(uint256).max);
    }
}
```

3 Appendix

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts

3.1.1 Metrics

In order to objectively assess the severity level of vulnerabilities in blockchain systems, this report provides detailed assessment metrics for security vulnerabilities in smart contracts with reference to CVSS 3.1 (Common Vulnerability Scoring System Ver 3.1).

According to the severity level of vulnerability, the vulnerabilities are classified into four levels: "critical", "high", "medium" and "low". It mainly relies on the degree of impact and likelihood of exploitation of the vulnerability, supplemented by other comprehensive factors to determine of the severity level.

Impact / Likelihood	Severe	High	Medium	Low
Probable	Critical	High	Medium	Low
Possible	High	High	Medium	Low
Unlikely	Medium	Medium	Low	Info
Rare	Low	Low	Info	Info

3.1.2 Degree of impact

- **Severe**

Severe impact generally refers to the vulnerability can have a serious impact on the confidentiality, integrity, availability of smart contracts or their economic model, which can cause substantial economic losses to the contract business system, large-scale data disruption, loss of authority management, failure of key functions, loss of credibility, or indirectly affect the operation of other smart contracts associated with it and cause substantial losses, as well as other severe and mostly irreversible harm.

- **High**

High impact generally refers to the vulnerability can have a relatively serious impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a greater economic loss, local functional unavailability, loss of credibility and other impact to the contract business system.

- **Medium**

Medium impact generally refers to the vulnerability can have a relatively minor impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a small amount of economic loss to the contract business system, individual business unavailability and other impact.

- **Low**

Low impact generally refers to the vulnerability can have a minor impact on the smart contract, which can pose certain security threat to the contract business system and needs to be improved.

3.1.4 Likelihood of Exploitation

- **Probable**

Probable likelihood generally means that the cost required to exploit the vulnerability is low, with no special exploitation threshold, and the vulnerability can be triggered consistently.

- **Possible**

Possible likelihood generally means that exploiting such vulnerability requires a certain cost, or there are certain conditions for exploitation, and the vulnerability is not easily and consistently triggered.

- **Unlikely**

Unlikely likelihood generally means that the vulnerability requires a high cost, or the exploitation conditions are very demanding and the vulnerability is highly difficult to trigger.

- **Rare**

Rare likelihood generally means that the vulnerability requires an extremely high cost or the conditions for exploitation are extremely difficult to achieve.

3.1.5 Fix Results Status

Status	**Description**
Fixed	The project party fully fixes a vulnerability.
Partially Fixed	The project party did not fully fix the issue, but only mitigated the issue.
Acknowledged	The project party confirms and chooses to ignore the issue.

3.2 Audit Categories

No.	Categories	Subitems
1	Coding Conventions	Compiler Version Security
		Deprecated Items
		Redundant Code
		require/assert Usage
		Gas Consumption
2	General Vulnerability	Integer Overflow/Underflow
		Reentrancy
		Pseudo-random Number Generator (PRNG)
		Transaction-Ordering Dependence
		DoS (Denial of Service)
		Function Call Permissions
		call/delegatecall Security
		Returned Value Security
		tx.origin Usage
		Replay Attack
		Overriding Variables
		Third-party Protocol Interface Consistency
3	Business Security	Business Logics
		Business Implementations
		Manipulable Token Price
		Centralized Asset Control
		Asset Tradability
		Arbitrage Attack

Beosin classified the security issues of smart contracts into three categories: Coding Conventions, General Vulnerability, Business Security. Their specific definitions are as follows:

- **Coding Conventions**

Audit whether smart contracts follow recommended language security coding practices. For example, smart contracts developed in Solidity language should fix the compiler version and do not use deprecated keywords.

- **General Vulnerability**

General Vulnerability include some common vulnerabilities that may appear in smart contract projects. These vulnerabilities are mainly related to the characteristics of the smart contract itself, such as integer overflow/underflow and denial of service attacks.

- **Business Security**

Business security is mainly related to some issues related to the business realized by each project, and has a relatively strong pertinence. For example, whether the lock-up plan in the code match the white paper, or the flash loan attack caused by the incorrect setting of the price acquisition oracle.

*Note that the project may suffer stake losses due to the integrated third-party protocol. This is not something Beosin can control. Business security requires the participation of the project party. The project party and users need to stay vigilant at all times.

3.3 Disclaimer

The Audit Report issued by Beosin is related to the services agreed in the relevant service agreement. The Project Party or the Served Party (hereinafter referred to as the "Served Party") can only be used within the conditions and scope agreed in the service agreement. Other third parties shall not transmit, disclose, quote, rely on or tamper with the Audit Report issued for any purpose.

The Audit Report issued by Beosin is made solely for the code, and any description, expression or wording contained therein shall not be interpreted as affirmation or confirmation of the project, nor shall any warranty or guarantee be given as to the absolute flawlessness of the code analyzed, the code team, the business model or legal compliance.

The Audit Report issued by Beosin is only based on the code provided by the Served Party and the technology currently available to Beosin. However, due to the technical limitations of any organization, and in the event that the code provided by the Served Party is missing information, tampered with, deleted, hidden or subsequently altered, the audit report may still fail to fully enumerate all the risks.

The Audit Report issued by Beosin in no way provides investment advice on any project, nor should it be utilized as investment suggestions of any type. This report represents an extensive evaluation process designed to help our customers improve code quality while mitigating the high risks in blockchain.

3.4 About Beosin

Beosin is the first institution in the world specializing in the construction of blockchain security ecosystem. The core team members are all professors, postdocs, PhDs, and Internet elites from world-renowned academic institutions. Beosin has more than 20 years of research in formal verification technology, trusted computing, mobile security and kernel security, with overseas experience in studying and collaborating in project research at well-known universities. Through the security audit and defense deployment of more than 2,000 smart contracts, over 50 public blockchains and wallets, and nearly 100 exchanges worldwide, Beosin has accumulated rich experience in security attack and defense of the blockchain field, and has developed several security products specifically for blockchain.