BEOSIN
Web3 Security & Compliance

FDGC

Smart Contract Security Audit

No. 202507291027

Jul 29th, 2025

SECURING BLOCKCHAIN ECOSYSTEM

WWW.BEOSIN.COM

Contents

Summary of Audit Results

After auditing, 1 Medium, 3 Low and 2 Info risk items were identified in the FDGC project. Specific audit details will be presented in the Findings section. Users should pay attention to the following aspects when interacting with this project:

- **Project Description:**

1. Basic Token Information

Token name	Fintech Gold Coin
Token symbol	FDGC
Decimals	6
Total supply	315,000 (Constant supply)
Token type	ERC-20

Table 1 FDGC token info

2. Business overview:

The FintechGoldCoin contract is an ERC-20 token ("FDGC", 6 decimals, total supply 315,000) featuring pausable transfers, and ownership transfer through a propose-and-claim mechanism. It includes an asset protection role to freeze/unfreeze addresses or wipe their balances (preventing frozen addresses from transferring or authorizing, with the option to destroy all tokens in frozen addresses via `wipeFrozenAddress`). It also supports a whitelist for delegated transfers using EIP712 signatures and a fee system managed by a fee controller to set transfer fee rates and recipients. The contract is designed for proxy-based upgrades.

1 Overview

1.1 Project Overview

Project Name	FDGC
Project Language	Solidity
Platform	BSC
Github Link	https://github.com/proofofash264/FDGC-contract (origin) https://github.com/vaibhavgarg18/FintechGoldCoin (Latest)
Commit	9da47d4b3009014c35d4efb406419c2c1b4f5d29 (origin) ed0bd1283323e26184ae5fa3d3c4eacc781c5614 (Latest)

1.2 Audit Overview

Audit work duration: Jul 25, 2025, Jul 29, 2025

Audit team: Beosin Security Team

1.3 Audit Method

The audit methods are as follows:

1. Formal Verification

Formal verification is a technique that uses property-based approaches for testing and verification. Property specifications define a set of rules using Beosin's library of security expert rules. These rules call into the contracts under analysis and make various assertions about their behavior. The rules of the specification play a crucial role in the analysis. If the rule is violated, a concrete test case is provided to demonstrate the violation.

2. Manual Review

Using manual auditing methods, the code is read line by line to identify potential security issues. This ensures that the contract's execution logic aligns with the client's specifications and intentions, thereby safeguarding the accuracy of the contract's business logic.

The manual audit is divided into three groups to cover the entire auditing process:

The Basic Testing Group is primarily responsible for interpreting the project's code and conducting comprehensive functional testing.

The Simulated Attack Group is responsible for analyzing the audited project based on the collected historical audit vulnerability database and security incident attack models. They identify potential attack vectors and collaborate with the Basic Testing Group to conduct simulated attack tests.

The Expert Analysis Group is responsible for analyzing the overall project design, interactions with third parties, and security risks in the on-chain operational environment. They also conduct a review of the entire audit findings.

3. Static Analysis

Static analysis is a function of examining code during compilation or static analysis to detect issues. Beosin-VaaS can detect more than 100 common smart contract vulnerabilities through static analysis, such as reentrancy and block parameter dependency. It allows early and efficient discovery of problems to improve code quality and security.

2 Findings

Index	Risk description	Severity level	Status
FDGC-01	Nonce Invalidation and Signature Malleability Attacks	Medium	Fixed
FDGC-02	Ambiguous Signer Recovery for from Address	Low	Fixed
FDGC-03	Unlimited Minting & Fee Cap Risk	Low	Fixed
FDGC-04	Centralization Risk	Low	Fixed
FDGC-05	Redundant Code	Info	Fixed
FDGC-06	Unrestricted Access to Domain Separator Initialization	Info	Fixed

Finding Details:

[FDGC-01] Nonce Invalidation and Signature Malleability Attacks

Severity Level	Medium
Lines	FintechGoldCoin.sol #L844-922
Type	General Vulnerability
Description	In the `executeMetaTransaction` function, the nonce is updated using nonces[userAddress] = nonces[userAddress]++. However, due to how post-increment works in Solidity, this assignment effectively stores the original nonce value, leaving it unchanged. As a result, the same meta-transaction signature can be replayed indefinitely, posing a serious replay attack risk. And verify in `BasicMetaTransaction` does not include the signer in the hash and does not prevent signature malleability attack.
Recommendation	It is recommended to modify the incorrect signature design or directly delete the `executeMetaTransaction` function.

```
 function executeMetaTransaction(
        address userAddress,
        bytes memory functionSignature,
        bytes32 sigR,
        bytes32 sigS,
        uint8 sigV
    ) public returns (bytes memory) {
...
        nonces[userAddress] = nonces[userAddress]++;
    function verify( address owner, uint256 nonce, uint256 chainID, bytes
memory functionSignature, bytes32 sigR, bytes32 sigS, uint8 sigV ) public
view returns (bool) {
        bytes32 hash = prefixed(
            keccak256(abi.encodePacked(nonce, this, chainID,
functionSignature))
        );
        address signer = ecrecover(hash, sigV, sigR, sigS);
        require(signer != address(0), "Invalid signature");
        return (owner == signer);
    }
```

Status

Fixed. Modified the assignment of nonce and added corresponding measures for signature ductility in verify.

```
        // nonces[userAddress] = nonces[userAddress]++;
        nonces[userAddress] += 1;
function verify(......

    ) public view returns(bool) {

        require(

            uint256(sigS) <=
0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A0,

            "Invalid signature 's' value"

        );

        require(sigV == 27 || sigV == 28, "Invalid signature 'v' value");
```

[FDGC-02] Ambiguous Signer Recovery for from Address

Severity Level	Low
Lines	FintechGoldCoin.sol #L1569-1598
Type	Business Security
Description	The `_betaDelegatedTransfer` function recovers the `from` address from an off-chain signature using `ecrecover(hash, v, r, s)`, but this recovered address is not explicitly verified against any user-supplied from parameter (in fact, `from` is implicitly extracted and hidden). This may lead to confusion in reasoning about who is actually authorizing the transfer, especially if used with interfaces or off-chain components that reference a from field. Moreover, improper signature generation or reuse across contexts could introduce replay or spoofing risks.

```
bytes32 hash = keccak256(
          abi.encodePacked(
              EIP191_HEADER,
              EIP712_DOMAIN_HASH,
              keccak256(
                  abi.encodePacked( // solium-disable-line
                          EIP712_DELEGATED_TRANSFER_SCHEMA_HASH,
                          bytes32(keccak256(abi.encodePacked(to))),
                          value,
                          serviceFee,
                          seq,
                          deadline
                      )
              )
          )
      );
      address _from = ecrecover(hash, v, r, s);
```

Recommendation	It is recommended to explicitly include the `from` address as an input in the signed message structure and validate that the recovered signer matches the expected address.
Status	**Fixed.** Added `from` in the signature data.

```
      bytes32 hash = keccak256(
          abi.encodePacked(
```

```
            EIP191_HEADER,
            EIP712_DOMAIN_HASH,
            keccak256(
                abi.encodePacked(
                    EIP712_DELEGATED_TRANSFER_SCHEMA_HASH,
                    bytes32(keccak256(abi.encodePacked(from))),
                    bytes32(keccak256(abi.encodePacked(to))),
                    value,
                    serviceFee,
                    seq,
                    deadline
                )
```

[FDGC-03] Unlimited Minting & Fee Cap Risk

Severity Level	Low
Lines	FintechGoldCoin.sol #L1756-1761
Type	Business Security
Description	The `supplyController` can mint tokens without an upper limit, posing a risk of uncontrolled token issuance. Additionally, the transaction fee cap can be set as high as 100%, potentially allowing the owner to extract the entire value of transactions.

```
    function increaseSupply(
        uint256 _value
    ) public onlySupplyController returns (bool success) {
        totalSupply_ = totalSupply_.add(_value);
        balances[supplyController] =
balances[supplyController].add(_value);
        emit SupplyIncreased(supplyController, _value);
        emit Transfer(address(0), supplyController, _value);
        return true;
    }
    function setFeeRate(uint256 _newFeeRate) public onlyFeeController
{
        require(_newFeeRate <= feeParts, "cannot set fee rate above
100%");
        uint256 _oldFeeRate = feeRate;
        feeRate = _newFeeRate;
        emit FeeRateSet(_oldFeeRate, feeRate);
    }
```

Recommendation	It is recommended to increase the token minting limit and lower the transaction fee limit.
Status	**Fixed.** The total amount of tokens was changed to 315,000, which were held by the owner during deployment, and minting and destruction were deleted. The upper limit of the handling fee has been modified.

```
        // Set a fixed total supply of 315,000 tokens
        uint256 initialSupply = 315000 * (10 ** uint256(decimals));
        totalSupply_ = initialSupply;
```

```
        // Assign all tokens to the owner's balance
        balances[_owner] = initialSupply;
    function setFeeRate(uint256 _newFeeRate) public onlyFeeController
{
        require(_newFeeRate <= 50000, "Fee rate cannot exceed 5%");
```

[FDGC-04] Centralization Risk

Severity Level	Low
Lines	FintechGoldCoin.sol
Type	Business Security
Description	The contract's owner holds the authority to pause the contract, modify minting processes, and freeze specific addresses. Frozen addresses are restricted from authorizing transactions or transferring funds, and their account balances can be entirely wiped using the `wipeFrozenAddress` function. Additionally, the contract employs a proxy pattern, which introduces risks such as potential vulnerabilities in the proxy implementation, upgradeability issues, or unauthorized access if the proxy's governance is not securely managed.
Recommendation	It is recommended to use a multi-signature wallet to manage special permission addresses such as owner.
Status	**Fixed.** The project owner stated that it will be managed using the method mentioned in the white paper.

[FDGC-05] Redundant Code

Severity Level	Info
Lines	FintechGoldCoin.sol #L1164
Type	Coding Conventions
Description	`feeDecimals` is not used anywhere else and is redundant code. `uint8 public constant feeDecimals = 6;`
Recommendation	It is recommended to delete the redundant code.
Status	**Fixed.**

[FDGC-06] Unrestricted Access to Domain Separator Initialization

Severity Level	Info
Lines	FintechGoldCoin.sol #L1266-1275
Type	Business Security
Description	The `initializeDomainSeparator` function is declared as public, allowing anyone to call it and overwrite the EIP712_DOMAIN_HASH at any time after contract deployment.

```
    function initializeDomainSeparator() public {
       // hash the name context with the contract address
       EIP712_DOMAIN_HASH = keccak256(
           abi.encodePacked( // solium-disable-line
                   EIP712_DOMAIN_SEPARATOR_SCHEMA_HASH,
                   keccak256(bytes(name)),
                   bytes32(keccak256(abi.encodePacked(address(this))
))
    }
```

Recommendation	It is recommended to change the visibility of the `initializeDomainSeparator` function to private or internal to prevent unauthorized or unintended updates to the domain separator.
Status	**Fixed.** Changed to `internal`.

```
function initializeDomainSeparator() internal {
       EIP712_DOMAIN_HASH = keccak256(
           abi.encodePacked(
               EIP712_DOMAIN_SEPARATOR_SCHEMA_HASH,
               keccak256(bytes(name)),
               bytes32(keccak256(abi.encodePacked(address(this))))
    }
```

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.

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

3.1.2 Degree of impact

- **Critical**

Critical 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.

3.1.3 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.

3.1.4 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.

BEOSIN
Web3 Security & Compliance

Official Website
https://www.beosin.com

Telegram
https://t.me/beosin

X
https://x.com/Beosin_com

Email
service@beosin.com

LinkedIn
https://www.linkedin.com/company/beosin/