SMART CONTRACT AUDIT REPORT

for

GCOW Token

Prepared By: Xiaomi Huang

PeckShield

November 21, 2025

Document Properties

Client	GCOW
Title	Smart Contract Audit Report
Target	GCOW
Version	1.0
Author	Xuxian Jiang
Auditors	Matthew Jiang, Xuxian Jiang
Reviewed by	Xiaomi Huang
Approved by	Xuxian Jiang
Classification	Public

Version Info

Version	Date	Author(s)	Description
1.0	November 21, 2025	Xuxian Jiang	Final Release
1.0-rc1	November 20, 2025	Xuxian Jiang	Release Candidate #1

Contact

For more information about this document and its contents, please contact PeckShield Inc.

Name	Xiaomi Huang
Phone	+86 156 0639 2692
Email	contact@peckshield.com

Contents

1 Introduction

Given the opportunity to review the design document and related source code of the GCOW token contract, we outline in the report our systematic method to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistency between smart contract code and the documentation, and provide additional suggestions or recommendations for improvement. Our analysis shows that the given version of the smart contracts can be further improved due to the presence of certain issues related to ERC20-compliance, security, or performance. This document outlines our audit results.

1.1 About GCOW

GCOW is the core token of GameCow, the third ecosystem of the COW.CM platform, with a fixed total supply of only 10,000 tokens, giving it extreme scarcity. GameCow aims to build a world-leading Web3 gaming aggregation platform, where all games use GCOW as the unified participation token. Users can continuously earn GCOW rewards through Web3 gameplay, black-hole encounters, on-chain LP addition, referral binding, and participation across the entire COW.CM ecosystem. With its dual mechanism of non-mintable supply + automatic burn, GCOW continually decreases in circulating supply, maintains long-term scarcity, and preserves strong growth potential, forming a more robust, stable, and sustainable game economy. The basic information of the audited contracts is as follows:

Table 1.1: Basic Information of GCOW Token

Item	Description
Name	GCOW
Website	https://cow.cm/
Type	ERC20 Token Contract
Platform	Solidity
Audit Method	Whitebox
Latest Audit Report	November 21, 2025

In the following, we show the deployment addresses of the audited token contract and the related

DividendTracker contract.[1]

- GCOW: https://bscscan.com/address/0xc95CF0286bB336031bB536856823bEbCDE67B299
- DividendTracker: https://bscscan.com/address/0x50969897c50808fc1d88A4bA5eeA6980125c87A8

And here is the deployment address of the audited token contract after all fixes for the issues found in the audit have been addressed:

- GCOW: https://bscscan.com/address/0xBB4fBE7E80f1D4c0e06428f80b4F26d9EE24B5D0

1.2 About PeckShield

PeckShield Inc. [9] is a leading blockchain security company with the goal of elevating the security, privacy, and usability of current blockchain ecosystem by offering top-notch, industry-leading services and products (including the service of smart contract auditing). We are reachable at Telegram (https://t.me/peckshield), Twitter (http://twitter.com/peckshield), or Email (contact@peckshield.com).

1.3 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [8]:

- Likelihood represents how likely a particular vulnerability is to be uncovered and exploited in the wild;
- Impact measures the technical loss and business damage of a successful attack;
- Severity demonstrates the overall criticality of the risk;

Likelihood and impact are categorized into three ratings: *H*, *M* and *L*, i.e., *high*, *medium* and *low* respectively. Severity is determined by likelihood and impact and can be classified into four categories accordingly, i.e., *Critical*, *High*, *Medium*, *Low* shown in Table 1.2.

We perform the audit according to the following procedures:

- Basic Coding Bugs: We first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.

[1] Note the associated Pool contract is not part of the audit.

Table 1.2: Vulnerability Severity Classification

Impact			
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low
	High	*Medium*	*Low*
	Likelihood		

- ERC20 Compliance Checks: We then manually check whether the implementation logic of the audited smart contract(s) follows the standard ERC20 specification and other best practices.
- Additional Recommendations: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

To evaluate the risk, we go through a list of check items and each would be labeled with a severity category. For one check item, if our tool does not identify any issue, the contract is considered safe regarding the check item. For any discovered issue, we might further deploy contracts on our private testnet and run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.3.

1.4 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release, and does not give any warranties on finding all possible security issues of the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contract(s). Last but not least, this security audit should not be used as investment advice.

Table 1.3: The Full List of Check Items

Category	Check Item
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	Money-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead of Transfer
	Costly Loop
	(Unsafe) Use of Untrusted Libraries
	(Unsafe) Use of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses
	Approve / TransferFrom Race Condition
ERC20 Compliance Checks	Compliance Checks (Section 3)
Additional Recommendations	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

2 Findings

2.1 Summary

Here is a summary of our findings after analyzing the `GCOW` token contract. During the first phase of our audit, we study the smart contract source code and run our in-house static code analyzer through the codebase. The purpose here is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by our tool. We further manually review business logics, examine system operations, and place ERC20-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

Severity	# of Findings	
Critical	0	
High	0	
Medium	1	■
Low	3	■ ■ ■
Informational	0	
Total	4	

Moreover, we explicitly evaluate whether the given contracts follow the standard ERC20 specification and other known best practices, and validate its compatibility with other similar ERC20 tokens and current DeFi protocols. The detailed ERC20 compliance checks are reported in Section 3. After that, we examine a few identified issues of varying severities that need to be brought up and paid more attention to. (The findings are categorized in the above table.) Additional information can be found in the next subsection, and the detailed discussions are in Section 4.

2.2 Key Findings

Overall, no ERC20 compliance issue was found and our detailed checklist can be found in Section 3. While there is no critical or high severity issue, the implementation can be improved by resolving the identified issues (shown in Table 2.1), including 1 medium-severity vulnerability and 3 low-severity vulnerabilities.

Table 2.1: Key GCOW Audit Findings

ID	Severity	Title	Category	Status
PVE-001	Low	Revisited Fee Amount Calculation Logic in _tokenTransfer()	Business Logic	Resolved
PVE-002	Low	Improved _transfer() Logic For Intended ERC20 Functionality	Coding Practices	Resolved
PVE-003	Low	Improved Validation of Function Arguments	Coding Practices	Resolved
PVE-004	Medium	Trust Issue Of Admin Keys	Security Features	Mitigated

Besides recommending specific countermeasures to mitigate the above issue(s), we also emphasize that it is always important to develop necessary risk-control mechanisms and make contingency plans, which may need to be exercised before the mainnet deployment. The risk-control mechanisms need to kick in at the very moment when the contracts are being deployed in mainnet. Please refer to Section 3 for our detailed compliance checks and Section 4 for elaboration of reported issues.

3 | ERC20 Compliance Checks

The ERC20 specification defines a list of API functions (and relevant events) that each token contract is expected to implement (and emit). The failure to meet these requirements means the token contract cannot be considered to be ERC20-compliant. Naturally, as the first step of our audit, we examine the list of API functions defined by the ERC20 specification and validate whether there exist any inconsistency or incompatibility in the implementation or the inherent business logic of the audited contract(s).

Table 3.1: Basic `View-Only` Functions Defined in The ERC20 Specification

Item	Description	Status
name()	Is declared as a public view function	✓
	Returns a string, for example "Tether USD"	✓
symbol()	Is declared as a public view function	✓
	Returns the symbol by which the token contract should be known, for example "USDT". It is usually 3 or 4 characters in length	✓
decimals()	Is declared as a public view function	✓
	Returns decimals, which refers to how divisible a token can be, from 0 (not at all divisible) to 18 (pretty much continuous) and even higher if required	✓
totalSupply()	Is declared as a public view function	✓
	Returns the number of total supplied tokens, including the total minted tokens (minus the total burned tokens) ever since the deployment	✓
balanceOf()	Is declared as a public view function	✓
	Anyone can query any address' balance, as all data on the blockchain is public	✓
allowance()	Is declared as a public view function	✓
	Returns the amount which the spender is still allowed to withdraw from the owner	✓

Our analysis shows that there is no ERC20 inconsistency or incompatibility issue found in the audited `GCOW` token contract.[1] In the surrounding two tables, we outline the respective list of basic `view-only` functions (Table 3.1) and key `state-changing` functions (Table 3.2) according to the widely-adopted ERC20 specification.

Table 3.2: Key `State-Changing` Functions Defined in The ERC20 Specification

Item	Description	Status
transfer()	Is declared as a public function	✓
	Returns a boolean value which accurately reflects the token transfer status	✓
	Reverts if the caller does not have enough tokens to spend	✓
	Allows zero amount transfers	✓
	Emits Transfer() event when tokens are transferred successfully (include 0 amount transfers)	✓
	Reverts while transferring to zero address	✓
transferFrom()	Is declared as a public function	✓
	Returns a boolean value which accurately reflects the token transfer status	✓
	Reverts if the spender does not have enough token allowances to spend	✓
	Updates the spender's token allowances when tokens are transferred successfully	✓
	Reverts if the from address does not have enough tokens to spend	✓
	Allows zero amount transfers	✓
	Emits Transfer() event when tokens are transferred successfully (include 0 amount transfers)	✓
	Reverts while transferring from zero address	✓
	Reverts while transferring to zero address	✓
approve()	Is declared as a public function	✓
	Returns a boolean value which accurately reflects the token approval status	✓
	Emits Approval() event when tokens are approved successfully	✓
	Reverts while approving to zero address	✓
Transfer() event	Is emitted when tokens are transferred, including zero value transfers	✓
	Is emitted with the from address set to $address(0x0)$ when new tokens are generated	✓
Approval() event	Is emitted on any successful call to approve()	✓

In addition, we perform a further examination on certain features that are permitted by the ERC20 specification or even further extended in follow-up refinements and enhancements, but not required for implementation. These features are generally helpful, but may also impact or bring certain incompatibility with current DeFi protocols. Therefore, we consider it is important to highlight them as well. This list is shown in Table 3.3.

[1]It is worth mentioning that the `BEP20` specification has defined an extended method of `EIP20`, i.e., `getOwner()`, which is currently not defined. Tokens that do not implement this method will not be able to flow across the `Binance Chain` and `Binance Smart Chain` (`BSC`). However, it does not affect normal `ERC20` functionalities.

Table 3.3: Additional `Opt-in` Features Examined in Our Audit

Feature	Description	Opt-in
Deflationary	Part of the tokens are burned or transferred as fee while on transfer()/transferFrom() calls	✓
Rebasing	The balanceOf() function returns a re-based balance instead of the actual stored amount of tokens owned by the specific address	—
Pausable	The token contract allows the owner or privileged users to pause the token transfers and other operations	—
Upgradable	The token contract allows for future upgrades	—
Whitelistable	The token contract allows the owner or privileged users to whitelist a specific address such that only token transfers and other operations related to that address are allowed	✓
Mintable	The token contract allows privileged account(s) to mint tokens to a specific address while maintaining overall total supply across supported chains	—
Burnable	The token contract allows privileged account(s) to burn tokens of a specific address while maintaining overall total supply across supported chains	—

4 Detailed Results

4.1 Revisited Fee Amount Calculation Logic in _tokenTransfer()

- ID: PVE-001
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: `GCOW`
- Category: Business Logic [7]
- CWE subcategory: CWE-841 [4]

Description

As mentioned earlier, the `GCOW` token is a deflationary one and certain fees are automatically deducted and collected for various purposes. While examining the logic to deduct the `LP` removal fees, we notice current implementation may be improved when calculating the fee amount for collection.

In the following, we show the code snippet from the related `_tokenTransfer()` routine. When the token transfer is part of liquidity removal operation, we notice the fee amount calculation differs in two different scenarios. The first scenario occurs when `userInfo.preLpAmount` is larger than current liquidity amount for removal and the fee amount is computed as `removeFeeAmt = isWithinRemoveLpBurnDays ? tAmount : 0;` (line 901). Otherwise, the second scenario involves the fee amount in two components: `removeLpBurnAmount` (line 905) and `removeLpFeeAmount` (line 914).

Our analysis shows current logic works fine under the assumption that the `feeAmount` variable initially holds `0`. However, this assumption is better avoided by ensuring the fee amount calculation is always sound. For example, the first scenario can adjust its calculation as `removeFeeAmt = isWithinRemoveLpBurnDays ? tAmount - feeAmount : 0;`. And in the second scenario, `removeLpBurnAmount` and `removeLpFeeAmount` can be better later revised as `removeLpBurnAmount = min(tAmount - feeAmount, removeLpBurnAmount)` and `removeLpBurnAmount = min(tAmount - feeAmount, removeLpFeeAmount)`, respectively.

```
893        if (isRemove && !isExcludedFromFeeTransfer) {
```

```
            UserInfo storage userInfo = _userInfo[recipient];

            // already remove prelp burn
            bool isWithinRemoveLpBurnDays = isStartTrade() && block.timestamp <
                startTradeTimestamp + removeLpBurnInterval;

            if (userInfo.preLpAmount >= removeLPLiquidity) {
                userInfo.preLpAmount -= removeLPLiquidity;
                uint256 removeFeeAmt = isWithinRemoveLpBurnDays ? tAmount : 0;
                feeAmount += removeFeeAmt;
                _takeTransfer(sender, DEAD_ADDRESS, removeFeeAmt);
            } else {
                uint256 removeLpBurnAmount = (userInfo.preLpAmount * tAmount) /
                    removeLPLiquidity;
                if (isWithinRemoveLpBurnDays && removeLpBurnAmount > 0) {
                    feeAmount += removeLpBurnAmount;
                    _takeTransfer(sender, DEAD_ADDRESS, removeLpBurnAmount);
                }
                // uint256 restLp = removeLPLiquidity - userInfo.preLpAmount;
                userInfo.lpAmount -= (removeLPLiquidity - userInfo.preLpAmount);
                userInfo.preLpAmount = 0;
                uint256 _tempvalue = tAmount - (isWithinRemoveLpBurnDays ?
                    removeLpBurnAmount : 0);
                uint256 removeLpFeeAmount = (_tempvalue * removeLiquidityFee) / 10000;
                if (removeLpFeeAmount > 0) {
                    feeAmount += removeLpFeeAmount;
                    _takeTransfer(sender, DEAD_ADDRESS, removeLpFeeAmount);
                }
            }
            dividendTracker.setBalance(payable(recipient), userInfo.lpAmount + userInfo.
                preLpAmount);

            emit UserLpAmountChangedWhenRemoveLp(
                recipient,
                userInfo.preLpAmount,
                userInfo.lpAmount
            );
        }
```

Listing 4.1: `GCOW::_tokenTransfer()`

Recommendation Revise the above-mentioned routine to properly compute the fee amounts for collection.

Status This issue has been resolved by following the above suggestion.

4.2 Improved _transfer() Logic For Intended ERC20 Functionality

- ID: PVE-002
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: `GCOW`
- Category: Coding Practices [6]
- CWE subcategory: CWE-563 [3]

Description

Though there is a standardized ERC-20 specification, many token contracts may not strictly follow the specification or have additional functionalities beyond the specification. In the following, we examine the `_transfer()` routine in `GCOW` and notice a minor issue that may be inconsistent with the expected ERC-20 functionality.

In particular, we show the code snippet from the `_transfer()` routine. On its entry of `_transfer()`, there is a check (line 613), i.e., `to == DEAD_ADDRESS && !isExcludedFromFee[from] && msg.sender == tx.origin`. If the check succeeds, suppose the pool is currently paused, it returns `return` (line 618) without adjusting the balances of `from` and `to`. As a result, the transaction still proceeds successfully without being reverted. From the user perspective, the transfer operation is successful while the balances remain unchanged. This is not compliant with the expected ERC-20 functionality and may cause issues if not handled properly. In fact, we shall choose to revert or adjust the transfer balances for `from` and `to`.

```
613     if (to == DEAD_ADDRESS && !isExcludedFromFee[from] && msg.sender == tx.origin) {
614         require(address(pool) != address(0), "pool not set");
615         require(amount <= maxAmountToJoinLottery, "exceed max amount");
616
617         if (pool.paused()) {
618             return;
619         }
620
621         pool.processLotteryEntry(from, amount);
622
623         try pool.batchProcessLotteryResultsWhenBurn() {} catch { emit FailedSwap(1); }
624
625         _basicTransfer(from, DEAD_ADDRESS, amount);
626         return;
627     }
```

Listing 4.2: `GCOW::_transfer()`

Recommendation Revisit the above routine to ensure the balances are properly adjusted before returning.

Status This issue has been fixed by invoking `_basicTransfer(from, DEAD_ADDRESS, amount)` before the return.

4.3 Improved Validation of Function Arguments

- ID: PVE-003
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: `GCOW`
- Category: Coding Practices [6]
- CWE subcategory: CWE-1126 [1]

Description

DeFi protocols typically have a number of system-wide parameters that can be dynamically configured on demand. The `GCOW` token is no exception. Specifically, if we examine the token contract, it has defined a number of token-wide risk parameters, such as `invitorRewardPercentList` and `isExcludedFromFee`. In the following, we show a specific setter routines that allows for the changes of `invitorRewardPercentList`.

```
    function setInvitorRewardPercentList(
        uint256[] calldata newValue
    ) public onlyOwner {
        invitorRewardPercentList = new uint256[](newValue.length);
        for (uint256 i = 0; i < newValue.length; i++) {
            invitorRewardPercentList[i] = newValue[i];
        }
    }
```

Listing 4.3: `GCOW::invitorRewardPercentList()`

These parameters define various aspects of the token operation and maintenance and need to exercise extra care when configuring or updating them. Our analysis shows the update logic on these parameters can be improved by applying more rigorous sanity checks. Based on the current implementation, certain corner cases may lead to an undesirable consequence. For example, an unlikely mis-configuration of `invitorRewardPercentList` may make the overall fee larger than the transfer amount. In other words, we shall ensure the following requirement after the above setter routine, i.e., `require(getInvitorTotalShare()<= 10000 - buyRewardFee - sellRewardFee - transferFee )`.

Recommendation Validate any changes regarding these token-wide parameters to ensure they fall in an appropriate range.

Status The issue has been fixed by following the above suggestion.

4.4 Trust Issue of Admin Keys

- ID: PVE-004
- Severity: Medium
- Likelihood: Low
- Impact: Medium
- Target: GCOW
- Category: Security Features [5]
- CWE subcategory: CWE-287 [2]

Description

In the audited GCOW contract, there is a privileged owner account that plays a critical role in governing and regulating the token-wide operations (e.g., configure parameters and manage fee-exclusion accounts). Our analysis shows that the privileged account needs to be scrutinized. In the following, we examine the privileged account and the related privileged accesses in current contract.

```
    function startLP() external onlyOwner {
        require(0 == startLPTimestamp, "startedAddLP");
        startLPTimestamp = block.timestamp;
    }

    function stopLP() external onlyOwner {
        startLPTimestamp = 0;
    }

    function launch() external onlyOwner {
        require(0 == startTradeTimestamp, "already open");
        startTradeTimestamp = block.timestamp;
    }

    function setStartTradeTimestamp(uint256 newValue) external onlyOwner {
        require(newValue > block.timestamp, "new value must be greater than current
            timestamp");
        startTradeTimestamp = newValue;
    }

    function shutdown() external onlyOwner {
        require(startTradeTimestamp > 0, "already shutdown");
        startTradeTimestamp = 0;
    }

    function setGasForProcessing(uint256 newValue) external onlyOwner {
        require(newValue >= 300000 && newValue <= 2000000, "gas for processing must be
            between 300000 and 2000000");
        gasForProcessing = newValue;
    }

    function setMinValue2AddLp2GetDividendValue(uint256 newValue) external onlyOwner {
        minValue2AddLp2GetDividendValue = newValue;
    }

    function multiSetIsExcludedFromFee(
        address[] calldata addresses,
        bool status
    ) external onlyOwner {
        for (uint256 i; i < addresses.length; i++) {
            isExcludedFromFee[addresses[i]] = status;
        }
    }
```

Listing 4.4: Example Privileged Operations in `GCOW`

We emphasize that the privilege assignment is necessary and consistent with the token design with cross-chain support. However, it would be worrisome if the privileged account is a plain `EOA` account. Note that a multi-sig account could greatly alleviate this concern, though it is still far from perfect. Specifically, a better approach is to eliminate the administration key concern by transferring the role to a community-governed `DAO`. In the meantime, a timelock-based mechanism can also be considered as mitigation.

Recommendation Promptly transfer the privileged account to the intended `DAO`-like governance contract. All changed to privileged operations may need to be mediated with necessary timelocks. Eventually, activate the normal on-chain community-based governance life-cycle and ensure the intended trustless nature and high-quality distributed governance.

Status This issue has been mitigated as the team plans to use a multi-sig as its `owner`.

5 Conclusion

In this security audit, we have examined the GCOW token design and implementation. During our audit, we first checked all respects related to the compatibility of the ERC20 specification and other known ERC20 pitfalls/vulnerabilities. We then proceeded to examine other areas such as coding practices and business logics. Overall, although no critical level vulnerabilities were discovered, we identified a small number of issues that need to be promptly addressed. In the meantime, as disclaimed in Section 1.4, we appreciate any constructive feedbacks or suggestions about our findings, procedures, audit scope, etc.

References

[1] MITRE. CWE-1126: Declaration of Variable with Unnecessarily Wide Scope. https://cwe.mitre.org/data/definitions/1126.html.

[2] MITRE. CWE-287: Improper Authentication. https://cwe.mitre.org/data/definitions/287.html.

[3] MITRE. CWE-563: Assignment to Variable without Use. https://cwe.mitre.org/data/definitions/563.html.

[4] MITRE. CWE-841: Improper Enforcement of Behavioral Workflow. https://cwe.mitre.org/data/definitions/841.html.

[5] MITRE. CWE CATEGORY: 7PK - Security Features. https://cwe.mitre.org/data/definitions/254.html.

[6] MITRE. CWE CATEGORY: Bad Coding Practices. https://cwe.mitre.org/data/definitions/1006.html.

[7] MITRE. CWE CATEGORY: Business Logic Errors. https://cwe.mitre.org/data/definitions/840.html.

[8] OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.

[9] PeckShield. PeckShield Inc. https://www.peckshield.com.