Dragon Capsule
Audit Report

Fri Dec 26 2025

contact@bitslab.xyz https://twitter.com/scalebit_

ScaleBit

Dragon Capsule Audit Report

1 Executive Summary

1.1 Project Information

Description	GSCNX is a security-focused ERC20 token on the BNB Smart Chain, optimized for PancakeSwap V3 and V4. The protocol features dynamic taxation, anti-bot protections, and transaction limits to ensure market stability, with infrastructure supporting seamless router and bridge integration. The contract is officially deployed at 0x34975Bc5B7BFE7a542C1538dB0282A44B64e8D3E (TX: 0xd74b5203b2949d23dd57d4f2b3bd45904c4538849a4531cbbf463c5fc8820b8b).
Type	Token
Auditors	Fishmen,N0n3,Nebu1a
Timeline	Thu Dec 25 2025 - Thu Dec 25 2025
Languages	Solidity
Platform	Ethereum
Methods	Architecture Review, Unit Testing, Manual Review

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
GSCNX	GSCNX.sol	a1598e65c3cb92cc5bd096d0e6de303c100e1148

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	5	5	0
Centralization	1	1	0
Critical	0	0	0
Major	2	2	0
Medium	1	1	0
Minor	0	0	0
Informational	1	1	0

1.4 ScaleBit Audit Breakdown

ScaleBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow
- Number of rounding errors
- Unchecked External Call
- Unchecked CALL Return Values
- Functionality Checks
- Reentrancy
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic issues
- Gas usage
- Fallback function usage
- tx.origin authentication
- Replay attacks
- Coding style issues

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by Dragon Capsule to identify any potential issues and vulnerabilities in the source code of the Dragon Capsule smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 5 issues of varying severity, listed below.

ID	Title	Severity	Status
GSC-1	Missing Event Emission within `setAntiSnipeEnabled`	Informational	Fixed
GSC-2	Fee Evasion Vulnerability via Precision Loss	Major	Fixed
GSC-3	Centralization Risks	Centralization	Fixed
GSC-4	Inconsistent `maxWallet` Validation Logic in GSCNX	Major	Fixed
GSC-5	The Circumvention of Ownership Waiver Restrictions	Medium	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the Dragon Capsule Smart Contract :

1. Owner / Administrative Process
The Owner holds the highest privilege and manages the contract's lifecycle.

- **Initialization**: Sets `maxTxAmount` , `maxWallet` , and `taxWallet` (defaults to deployment address).
- **Configuration**: Whitelists `isLiquidityRouter` (e.g., V4 Vaults) and marks `automatedMarketMakerPairs` .
- **Launch**: Calls `enableTrading()` (irreversible). This starts the `tradingStartBlock` and `tradingStartTime` parameters.
- **Exit Strategy**: Can `renounceOwnership()` , but only **after** trading is active, ensuring the contract isn't left in a locked state.

2. User (Trader) Process
Users interact with the token primarily through decentralized exchanges (PancakeSwap).

- **Buying**: Subject to `initialBuyFee` (launch phase) or `FINAL_BUY_FEE` (5% after 15 mins).
- **Selling**: Currently subject to 0% tax (`FINAL_SELL_FEE`).
- **Compliance**: Transfers must respect `maxTxAmount` and `maxWallet` limits.
- **Anti-Snipe**: During the first 8 blocks post-launch, users face stricter transaction limits (1/5th of `maxTxAmount`).

3. Liquidity Infrastructure Process
Addresses marked as `isLiquidityRouter` or `automatedMarketMakerPairs` have unique roles.

- **Exempt Routers**: Bypasses fees and limits entirely. This is critical for V4 Hooks or cross-chain bridges where internal accounting (flash accounting) must not be interrupted by taxes.

- **AMM Pairs**: Act as the "trigger" for buy/sell logic. Transfers **from** a pair are flagged as buys; transfers **to** a pair (not from an exempt address) are flagged as sells.

4. Fee & Treasury Process

- **Collection**: On every taxable buy, the contract splits the transfer: `feeAmount` goes to the `taxWallet`, and the remainder goes to the user.
- **Decimals**: The contract uses 6 decimals.

4 Findings

GSC-1 Missing Event Emission within `setAntiSnipeEnabled`

Severity: Informational

Status: Fixed

Code Location:

GSCNX.sol#126-128

Descriptions:

The `setAntiSnipeEnabled` function in the `GSCNX` contract is used to update the `antiSnipeEnabled` state variable, which controls critical **anti-snipe mechanisms** and **transaction limit protection logic** within the contract. However, this function does **not emit any event** when modifying this critical configuration parameter.

The absence of an event emission prevents **off-chain monitoring systems**, **decentralized applications (dApps)**, and **community users** from performing real-time and intuitive monitoring of the anti-snipe protection switch's status changes. This lack of transparency increases the risk that **accidental or malicious configuration changes** made by the administrator go undetected and significantly reduces the system's **auditability** and **operational transparency**.

Suggestion:

It is recommended to introduce and emit a dedicated event (e.g., `AntiSnipeEnabledUpdated`) within the `setAntiSnipeEnabled` function. Emitting this event whenever the `antiSnipeEnabled` status changes will allow off-chain listeners to track critical configuration updates reliably and improve transparency for sensitive contract operations.

Resolution:

The team adopted our advice and fixed this issue by adding the `setAntiSnipeEnabled` event emission. The hash of the improved code is: a1598e65c3cb92cc5bd096d0e6de303c100e1148.

GSC-2 Fee Evasion Vulnerability via Precision Loss

Severity: Major

Status: Fixed

Code Location:

GSCNX.sol#234

Descriptions:

The `_update` function calculates the transaction fee using the formula `feeAmount = amount * fee / 10000`. With the `FINAL_BUY_FEE` configured at 500 (5%), this calculation is mathematically equivalent to `amount / 20`.

Due to Solidity's integer division mechanism, any transaction `amount` less than 20 results in a `feeAmount` of 0 (e.g., `19 * 500 / 10000 = 0`).

This precision loss creates a loophole that allows users to bypass the fee mechanism. An attacker could exploit this by splitting a large purchase into a massive number of micro-transactions, each with an amount less than 20 (e.g., 19). By executing these dust transactions repeatedly—potentially using multiple addresses to avoid rate limits—the attacker can accumulate tokens without paying the required 5% tax.

```
if (isBuy && buyTaxEnabled) {
            // Time-based dynamic fee
            if (block.timestamp > tradingStartTime + 900) {
                feeAmount = amount * FINAL_BUY_FEE / 10000; // 5%
            } else {
                feeAmount = amount * initialBuyFee / 10000; // Variable (Default 20%)
            }
        }
```

Suggestion:

It is recommended that the calculation logic for `feeAmount` be revised.

Resolution:

The team adopted our advice and fixed this issue by revising the rate calculation logic using the rounding up algorithm. The hash of the improved code is: a1598e65c3cb92cc5bd096d0e6de303c100e1148.

GSC-3 Centralization Risks

Severity: Centralization

Status: Fixed

Code Location:

GSCNX.sol

Descriptions:

The `GSCNX` contract grants the `owner` role extensive privileges to modify critical protocol parameters. Several centralization risks were identified in the protocol:

1. **Fee Configuration Control**: The owner can adjust the `initialBuyFee` up to 25% and toggle the `buyTaxEnabled` status. This allows the owner to significantly alter the cost of trading for users.
2. **Limit Adjustments**: The owner has the authority to modify `maxTxAmount` and `maxWallet` via `setLimits`. While there are lower bounds to prevent complete restriction, the owner can still tighten these limits to the minimum allowed values.
3. **Privileged Exemptions**: Through `setLiquidityRouter`, the owner can whitelist arbitrary addresses to bypass all fees and transaction limits. This power could be used to grant unfair advantages to specific users or wallets.
4. **Tax Destination Control**: The owner can arbitrarily change the `taxWallet` via `setTaxWallet`, controlling the flow of protocol revenue.
5. **Market Definition**: The owner determines which addresses are treated as automated market makers via `setAutomatedMarketMakerPair`, which directly affects which transactions are taxed.

Suggestion:

It is recommended to transfer the ownership to a multi-signature wallet or a DAO governance contract to mitigate the risk of single-point failure or malicious actions by a

single administrator. Additionally, consider using a timelock mechanism for sensitive parameter changes to give the community time to react to updates.

Resolution:

The team mitigated the centralization risks and adopted our recommendation by transferring ownership to a 3/5 Gnosis Safe Multi-Signature wallet (`0xCb179cf0703C2923f266316cEa0f1d7E4B76571C`) to mitigate single-point failures. To prevent collusion, the signer keys are distributed among independent representatives:

- **Signer 1 (Super Hub Representative)**: `0x129A127C0927999a927aE9654A2B1947f142673b`
- **Signer 2 (Security and Audit Representative)**: `0x49fe1ea9cbb1b70e0bf3a88f12ba245b63c3a7d6`
- **Signer 3 (Laboratory Representative)**: `0xe0e919aB841d42C8A3C4058788F3E1AC7111Cd02`
- **Signer 4 (Legal Compliance Representative)**: `0x333904c4bff4ef5bb9b78abf1841851dd646dc7e`
- **Signer 5 (Long-term Ecosystem Representative)**: `0xE99238c4Ff9968116FC71232BA09Cae8f36690b9`

Additionally, the team provided the following clarifications and measures:

1. **Fee Configuration**: The adjustable fee mechanism is a temporary **Anti-Snipe** measure restricted to the first 15 minutes of trading, after which the buy tax is automatically fixed at 5%.
2. **Privileged Exemptions**: The `setLiquidityRouter` function is strictly limited to officially recognized routers (e.g., PancakeSwap V4) for forward compatibility. Any modification requires **3/5 Multi-Signature execution** and adheres to the following protocols:
    - **Transparent Process**: All operations will be publicly disclosed in advance through official channels (Website, Twitter, Telegram) to ensure community verification.
    - **Security Assessment**: Before whitelisting, the team will carefully evaluate the target router contract to ensure its behavior is controllable and does not disrupt

existing fee and transaction control logic.

3. **Token Custody**: All undistributed tokens have been transferred to the Gnosis Safe Multi-Signature wallet for secure custody.

4. **Transparency & Governance**: The team has open-sourced the codebase at https://github.com/youtz88/GSCNX and disclosed the token allocation plan. The project is committed to a roadmap transitioning towards full decentralization, eventually renouncing ownership.

GSC-4 Inconsistent `maxWallet` Validation Logic in GSCNX

Severity: Major

Status: Fixed

Code Location:

GSCNX.sol#83,124

Descriptions:

The `GSCNX` contract appears to establish a design standard where `maxWallet` is 1% of the total supply. This is evidenced by the constructor initialization `maxWallet = (TOTAL_SUPPLY * 10) / 1000;` and the explicit comment `// Max Wallet: 1.0% of Total Supply`.
However, the validation logic in the `setLimits` function contains a likely calculation error or inconsistency. The line `require(_maxWallet >= TOTAL_SUPPLY / 1000, ...)` enforces a minimum limit of only 0.1% (`1/1000`), which is 10 times lower than the 1% (`10/1000`) established in the constructor. This discrepancy suggests that the developer may have missed a multiplication factor (e.g., `* 10`) when writing the validation logic. If left uncorrected, this allows the wallet limit to be set significantly lower than intended, potentially disrupting normal token usage.

Suggestion:

It is recommended to correct the validation logic in `setLimits` to `_maxWallet >= (TOTAL_SUPPLY * 10) / 1000` to align with the initial 1% limit design, as the current `TOTAL_SUPPLY / 1000` (0.1%) appears to be a calculation error. This correction will maintain consistency between the documented design intent and the enforced logic.

Resolution:

The team adopted our advice and fixed this issue by correcting the validation logic in setLimits to be consistent with the initial 1% limit design. The hash of the improved code is: a1598e65c3cb92cc5bd096d0e6de303c100e1148.

GSC-5 The Circumvention of Ownership Waiver Restrictions

Severity: Medium

Status: Fixed

Code Location:

GSCNX.sol

Descriptions:

The contract rewrites the `renounceOwnership` function inherited from the `OpenZeppelin Ownable` contract, restricting that ownership can only be relinquished after trading has been activated (`tradingActive == true`). This measure aims to prevent project owners from relinquishing control before the token is listed for trading. However, the contract does not rewrite the similarly inherited `transferOwnership` function. A malicious or negligent owner could call `transferOwnership(address(0))` , which has precisely the same effect as calling `renounceOwnership()` : transferring ownership to the zero address, thereby permanently relinquishing control. Since the `transferOwnership` function lacks corresponding checks, the restriction set by `renounceOwnership` can be easily circumvented, rendering this security measure ineffective.

Suggestion:

To enforce the restriction, the `transferOwnership(address newOwner)` function must be rewritten concurrently. In the new implementation, it is necessary to check whether `newOwner` is address(0). If true, the same `require(tradingActive, ...)` check applied to `renounceOwnership` should be implemented. This will ensure all paths for relinquishing ownership are subject to the same security policy constraints.

Resolution:

The team adopted our advice and fixed this issue by rewriting the `transferOwnership` function. The hash of the improved code is: a1598e65c3cb92cc5bd096d0e6de303c100e1148.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.