SALUS SECURITY

JAN 2024

CODE SECURITY ASSESSMENT

GREEN CANDLE

Overview

Project Summary

- Name: Green Candle (GC)
- Platform: Binance Smart Chain
- Address: 0x55e48AA00C55B6ed4453d90C247833F604bAc1CB
- Language: Solidity
- Repository:
    - https://github.com/GreenCandleman/Green-Candle
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Green Candle (GC)
Version	v2
Type	Solidity
Dates	Jan 04 2024
Logs	Dec 31 2023; Jan 04 2024

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	1
Total Low-Severity issues	1
Total informational issues	2
Total	4

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 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 with 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.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Spenders may spend more than token owners expected	Medium	Business Logic	Acknowledged
2	Centralization risk	Low	Business Logic	Acknowledged
3	Typo	Informational	Code Quality	Acknowledged
4	Compare bool variables to bool literals	Informational	Gas Optimization	Acknowledged

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Spenders may spend more than token owners expected

Severity: Medium	Category: Business Logic
Target: - CoinToken.sol	

Description

In the transferFrom() function, the spender can control the token owner's tokens with approval. So when the spender transfers the token from the token owner, in addition to modifying the balance of the token owner, it also needs to reduce the allowance with the same amount.

In the implementation, although the balance of _from is reduced by _value, the allowance is reduced by the _value after deducting txFee and burnFee. In this case, the actual spend amount could be larger than the allowance, if txFee or burnFee is non-zero.

CoinToken.sol:L192-L219

```
function transferFrom(address _from, address _to, uint256 _value) public returns (bool)
{
        ...
        balances[_from] = balances[_from].sub(_value);
        ...
        if(txFee > 0 && _from != FeeAddress){
            ...
            _value =  _value.sub(DenverDeflaionaryDecay);
        }

        if(burnFee > 0 && _from != FeeAddress){
            ...
            _value =  _value.sub(Burnvalue);
        }

        balances[_to] = balances[_to].add(_value);
        allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
        emit Transfer(_from, _to, _value);
        return true;
}
```

Proof of concept

```
function testTransferFrom() public {
        vm.startPrank(owner);
        // set fee
        token.updateFee(10, 10, alice);
        // approve
        token.approve(alice, token.balanceOf(owner));
        vm.stopPrank();
```

```
        vm.startPrank(alice);
        uint256 amount = uint256(1e12);
        console2.log("allowance before transfer:", token.allowance(owner, alice));
        console2.log("the amount transfer to bob:", amount);
        // transferFrom
        token.transferFrom(owner, bob, amount);
        console2.log("allowance after transfer:", token.allowance(owner, alice));
        require(token.allowance(owner, alice) > 4e12);
}
```

Recommendation

Consider reducing the spender's allowance before the fee calculation.

Status

This issue has been acknowledged by the team.

2. Centralization risk

Severity: Low	Category: Centralization
Target: - CoinToken.sol	

Description

The contract compiled with the 0.4.x version compiler lacks arithmetic overflow/underflow protection. Therefore, the addition operation of `block.timestamp + time` in the lock() function might overflow, causing _lockTime to become a small value, resulting in insufficient restrictions in the unlock() function.

CoinToken.sol:L64-L69

```
function lock(uint256 time) public onlyOwner {
        _previousOwner = owner;
        owner = address(0);
        _lockTime = block.timestamp + time;
        emit OwnershipTransferred(owner, address(0));
}
```

When the token is transferred, txFee will be transferred to FeeAddress and burnFee will be burned. The txFee, burnFee, and FeeAddress can be set by the owner with no upper limit. In addition, the owner can also pause token trading.

CoinToken.sol:L319-L323

```
function updateFee(uint256 _txFee,uint256 _burnFee,address _FeeAddress) onlyOwner
public{
        txFee = _txFee;
        burnFee = _burnFee;
        FeeAddress = _FeeAddress;
}
```

CoinToken.sol:L115-L118, L123-L126

```
function pause() onlyOwner whenNotPaused public {
        paused = true;
        emit Pause();
}

function unpause() onlyOwner whenPaused public {
        paused = false;
        emit Unpause();
}
```

Since the owner is an EOA account, if an attacker compromises the owner's private key, he/she can control the status of the contract by setting these variables.

Recommendation

It is recommended to transfer privileged accounts to multi-sig accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

Additionally, consider utilizing functions from SafeMath for arithmetic operations to prevent potential overflow/underflow issues in the code.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

3. Typo

Severity: Informational	Category: Code Quality
Target: - CoinToken.sol	

Description

DenverDeflaionaryDecay should be DenverDeflationaryDecay.

CoinToken.sol:L165-L170

```
if(txFee > 0 && msg.sender != FeeAddress){
        uint256 DenverDeflaionaryDecay = tempValue.div(uint256(100 / txFee));
        balances[FeeAddress] = balances[FeeAddress].add(DenverDeflaionaryDecay);
        emit Transfer(msg.sender, FeeAddress, DenverDeflaionaryDecay);
        _value =  _value.sub(DenverDeflaionaryDecay);
}
```

CoinToken.sol:L201-L206

```
if(txFee > 0 && _from != FeeAddress){
        uint256 DenverDeflaionaryDecay = tempValue.div(uint256(100 / txFee));
        balances[FeeAddress] = balances[FeeAddress].add(DenverDeflaionaryDecay);
        emit Transfer(_from, FeeAddress, DenverDeflaionaryDecay);
        _value =  _value.sub(DenverDeflaionaryDecay);
}
```

Recommendation

Consider fixing the typo.

Status

This issue has been acknowledged by the team.

4. Compare bool variables to bool literals

Severity: Informational	Category: Gas Optimization
Target: - CoinToken.sol	

Description

Boolean variables can be directly used in condition checks without comparing them explicitly to bool literals.

CoinToken.sol:L159-L160

```
require(tokenBlacklist[msg.sender] == false);
require(tokenBlacklist[_to] == false);
```

CoinToken.sol:L193-L195

```
require(tokenBlacklist[msg.sender] == false);
require(tokenBlacklist[_from] == false);
require(tokenBlacklist[_to] == false);
```

Recommendation

Consider using `!tokenBlacklist[user]` instead of `tokenBlacklist[user] == false`.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following file from address 0x55e48AA00C55B6ed4453d90C247833F604bAc1CB:

File	SHA-1 hash
CoinToken.sol	75134b0691a8d745a99488f01b44ef51646bc294