Crescite

Smart Contract Security Audit

Prepared by ShellBoxes

Aug 8th, 2023 - Aug 9th, 2023

Shellboxes.com

contact@shellboxes.com

Document Properties

Client	Crescite
Version	1.0
Classification	Public

Scope

Repository	Commit Hash
https://github.com/Crescite/crescite-token	a080b0bb8fa182bc3d80a628717d581e5ee10a97

Re-Audit

Repository	Commit Hash
https://github.com/Crescite/crescite-token	a080b0bb8fa182bc3d80a628717d581e5ee10a97

Contacts

COMPANY	EMAIL
ShellBoxes	contact@shellboxes.com

Contents

1 Introduction 4
1.1 About Crescite . . . 4
1.2 Approach & Methodology . . . 4
1.2.1 Risk Methodology . . . 5

2 Findings Overview 6
2.1 Summary . . . 6
2.2 Key Findings . . . 6

3 Finding Details 7
SHB.1 Inaccurate maxFlashLoan Value Due to Interaction Between ERC20FlashMint and ERC20Votes . . . 7
SHB.2 Unrestricted Control Over Supply by MINTER_ROLE . . . 8
SHB.3 Potential for Disruption of Transfers by PAUSER_ROLE . . . 9
SHB.4 Approve Race Condition . . . 10
SHB.5 Floating Pragma . . . 11

4 Best Practices 13
BP.1 Update OpenZeppelin contracts Version For Optimized ERC20Votes . . . 13

5 Conclusion 14

6 Scope Files 15
6.1 Audit . . . 15
6.2 Re-Audit . . . 15

7 Disclaimer 16

1 Introduction

Crescite engaged ShellBoxes to conduct a security assessment on the Crescite beginning on Aug 8th, 2023 and ending Aug 9th, 2023. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1 About Crescite

The Crescite Protocol is a Layer 2 built on the open-source XDC Network, allowing for a decentralized, hybrid, interoperable, and liquid network.The Crescite Token is an XRC-20 Token. The token is meant to be used for functional utility within our community platform to grow the ideals of a faith-based blockchain community and further ESG impact across the world.

Issuer	Crescite
Website	`https://www.crescite.org`
Type	Solidity Smart Contract
Whitepaper	`https://www.crescite.org/_files/ugd/fd64a1_eeca34ad5f6e4bb68701fceb9b7aeac7.pdf`
Audit Method	Whitebox

1.2 Approach & Methodology

ShellBoxes used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's scope. While manual testing is advised for identifying problems in logic, procedure, and

implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

1.2.1 Risk Methodology

Vulnerabilities or bugs identified by ShellBoxes are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.
- Impact quantifies the technical and economic costs of a successful attack.
- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.

Impact				
	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low
		High	Medium	Low

Likelihood

2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the Crescite implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.2 Key Findings

In general, these smart contracts are well-designed and constructed, but their implementation might be improved by addressing the discovered flaws, which include , 3 medium-severity, 2 low-severity vulnerabilities.

Vulnerabilities	Severity	Status
SHB.1. Inaccurate maxFlashLoan Value Due to Interaction Between ERC20FlashMint and ERC20Votes	MEDIUM	Acknowledged
SHB.2. Unrestricted Control Over Supply by MINTER_ROLE	MEDIUM	Acknowledged
SHB.3. Potential for Disruption of Transfers by PAUSER_ROLE	MEDIUM	Acknowledged
SHB.4. Approve Race Condition	LOW	Acknowledged
SHB.5. Floating Pragma	LOW	Acknowledged

3 Finding Details

SHB.1 Inaccurate maxFlashLoan Value Due to Interaction Between ERC20FlashMint and ERC20Votes

- Severity : MEDIUM
- Status : Acknowledged
- Likelihood : 1
- Impact : 3

Description:

The smart contract uses the ERC20FlashMint extension in conjunction with the ERC20Votes extension. This combination can lead to an inaccurate value for maxFlashLoan, which will not correctly reflect the maximum amount that can be flash minted. This discrepancy could lead to unexpected behavior when interacting with the flash mint function.

Files Affected:

SHB.1.1: Crescite.sol

```
10 import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Votes.sol";
11 import "@openzeppelin/contracts/token/ERC20/extensions/ERC20FlashMint.
       ↪ sol";
```

SHB.1.2: Crescite.sol

```
13 contract Crescite is ERC20, ERC20Burnable, ERC20Snapshot, AccessControl,
       ↪  Pausable, ERC20Permit, ERC20Votes, ERC20FlashMint {
```

Recommendation:

Consider overriding the maxFlashLoan function in the ERC20FlashMint contract to use the _maxSupply() function from ERC20Votes. This will ensure that maxFlashLoan accurately reflects the maximum amount that can be flash minted. The _maxSupply() function returns

the maximum supply value of type(uint224).max, which should be the correct maximum for flash minting in this context.

Updates

The team has acknowledged the issue.

SHB.2 Unrestricted Control Over Supply by MINTER_ROLE

- Severity : MEDIUM
- Likelihood : 1
- Status : Acknowledged
- Impact : 3

Description:

In the provided smart contract, the MINTER_ROLE has unrestricted control over the token supply. This role is granted the ability to mint an unlimited number of tokens. While this may be intended functionality, it poses a risk if the account with the MINTER_ROLE is compromised or misused. An attacker gaining control of this role could inflate the token supply, potentially devaluing the token and disrupting the token economy. Therefore, there are no guarantees that the token distribution will be performed as stated in the crescite-tokenomics.

Files Affected:

SHB.2.1: Crescite.sol

```
37 function mint(address to, uint256 amount) public onlyRole(MINTER_ROLE) {
38     _mint(to, amount);
39 }
```

Recommendation:

Consider implementing a mechanism to ensure that the token will be distributed as stated in the crescite-tokenomics.

Updates

The team has acknowledged the issue.

SHB.3 Potential for Disruption of Transfers by PAUSER_ROLE

- Severity : MEDIUM
- Status : Acknowledged
- Likelihood : 1
- Impact : 3

Description:

The smart contract includes a PAUSER_ROLE that has the ability to pause and unpause the contract. When the contract is paused, all token transfers are halted. This could potentially disrupt the token's economy if the PAUSER_ROLE is misused or falls into the wrong hands. An attacker with control of this role could halt all token transfers, causing significant disruption to users and potentially damaging trust in the token.

Files Affected:

SHB.3.1: Crescite.sol

```
29 function pause() public onlyRole(PAUSER_ROLE) {
30     _pause();
31 }
```

SHB.3.2: Crescite.sol

```
41 function _beforeTokenTransfer(address from, address to, uint256 amount)
42     internal
```

```
43      whenNotPaused
44      override(ERC20, ERC20Snapshot)
45  {
46      super._beforeTokenTransfer(from, to, amount);
47  }
```

Recommendation:

Consider implementing additional safeguards around the use of the PAUSER_ROLE. This could include requiring multiple signatures to pause the contract, or implementing a time delay before the pause function takes effect, giving users a chance to react.

Updates

The team has acknowledged the issue, stating that they will be assigning the role to a multisig wallet.

SHB.4 Approve Race Condition

- Severity : LOW
- Status : Acknowledged
- Likelihood : 1
- Impact : 2

Description:

The standard ERC20 implementation contains a well-known racing condition in its approve function, wherein a spender can observe the token owner broadcast a transaction altering their approval and then quickly sign and broadcast a transaction using transferFrom to claim the current approved amount to the spender's balance. If the spender's transaction is verified prior to the owner's, the spender will be able to receive both approval amounts.

Files Affected:

SHB.4.1: Crescite.sol

```
4 import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
```

SHB.4.2: Crescite.sol

```
13 contract Crescite is ERC20, ERC20Burnable, ERC20Snapshot, AccessControl,
       ↪ Pausable, ERC20Permit, ERC20Votes, ERC20FlashMint {
```

Recommendation:

We recommend using increaseAllowance and decreaseAllowance functions to modify the approval amount instead of using the approve function to modify it.

Updates

The team has acknowledged the issue.

SHB.5 Floating Pragma

- Severity : LOW
- Status : Acknowledged
- Likelihood : 1
- Impact : 1

Description:

The contract uses a floating pragma statement pragma solidity 0.8.17;. This means that the contract can be compiled with any compiler version from 0.8.17 (inclusive) up to, but not including, version 0.9.0. This could potentially introduce unexpected behavior if the contract is compiled with a newer compiler version that includes breaking changes.

Files Affected:

SHB.5.1: Crescite.sol

```
2 pragma solidity ^0.8.17;
```

Recommendation:

It is generally recommended to lock the pragma statement to a specific compiler version to ensure that the contract behaves as expected. This can be done by removing the caret from the pragma statement.

Updates

The team has acknowledged the issue.

4 Best Practices

BP.1 Update OpenZeppelin contracts Version For Optimized ERC20Votes

Description:

The smart contract currently uses OpenZeppelin version 4.8.0. However, version 4.9.0 or later is recommended due to an optimization update in the ERC20Votes contract. This update uses unchecked arithmetic operations to optimize gas costs. By continuing to use version 4.8.0, the contract may incur unnecessary and potentially significant gas costs. Consider upgrading the OpenZeppelin contracts to version 4.9.0 or later. This will include the optimization update in the ERC20Votes contract and could lead to reduced gas costs for contract interactions.

Files Affected:

BP.1.1: Crescite.sol

```
10 import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Votes.sol";
```

BP.1.2: package.json

```
21 "@openzeppelin/contracts": "^4.8.0",
```

Status - Acknowledged

5 Conclusion

In this audit, we examined the design and implementation of Crescite contract and discovered several issues of varying severity. Crescite team acknowledged 5 issues raised in the initial report classifying them as a risk with low-probability of occurrence. Shellboxes' auditors advised Crescite Team to maintain a high level of vigilance and to keep those findings in mind in order to avoid any future complications.

6 Scope Files

6.1 Audit

Files	MD5 Hash
contracts/Crescite.sol	f4f9034bba07f782917a9d5f55ffeddb

6.2 Re-Audit

Files	MD5 Hash
contracts/Crescite.sol	f4f9034bba07f782917a9d5f55ffeddb

7 Disclaimer

Shellboxes reports should not be construed as ”endorsements” or ”disapprovals” of particular teams or projects. These reports do not reflect the economics or value of any ”product” or ”asset” produced by any team or project that engages Shellboxes to do a security evaluation, nor should they be regarded as such. Shellboxes Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the examined technology, nor do they provide any indication of the technology's proprietors, business model, business or legal compliance. Shellboxes Reports should not be used in any way to decide whether to invest in or take part in a certain project. These reports don't offer any kind of investing advice and shouldn't be used that way. Shellboxes Reports are the result of a thorough auditing process designed to assist our clients in improving the quality of their code while lowering the significant risk posed by blockchain technology. According to Shellboxes, each business and person is in charge of their own due diligence and ongoing security. Shellboxes does not guarantee the security or functionality of the technology we agree to research; instead, our purpose is to assist in limiting the attack vectors and the high degree of variation associated with using new and evolving technologies.

For a Contract Audit, contact us at contact@shellboxes.com