PRISMA SHIELD

CRESCENTSWAP

DEEP LOGIC AUDIT REPORT

CrescentSwap: Moonlight token and MoonlightRedeem Contracts

MAY 24 2023

PRISMA SHIELD

Table of Contents

Overview

This audit only covers the Moonlight token and MoonlightRedeem contracts. It does not cover any other contracts built by CrescentSwap.

Moonlight token a simple ERC20 token with added centralization of control. It allows the contract owner (Gnosis multisig address 0x70fDFC034f2AB7Ab8E279f1A30d4Af2905F8C06D) to blacklist addresses from transferring the token. The reason the team gave for having this feature is to disallow the token from being listed on some DEXes. The token also allows the owner to disable token transfers in general.

MoonlightRedeem allows burning Moonlight tokens in exchange for receiving USDT and other tokens added to the MoonlightRedeem contract.

What is a Deep Logic Audit?

A deep logic smart contract audit is a human-driven code review that checks all of the code business logic for bugs, mathematical errors, and security risks. The audit verifies that the code honors the whitepaper. In addition, this service includes mainnet testing and proactive communication with the project owners to ensure full comprehension of the project to provide the best possible code review quality.

Overview

Findings Summary

	Total Findings	Resolved	Acknowledged
Total Findings	**13**	**13**	**0**
High Security Findings	3	3	0
Medium Security Findings	0	0	0
High Logical Findings	5	5	0
Medium Logical Findings	2	2	0
Informational Findings	3	3	0

ID	Section	Type	Severity	Page	Status
MLT-01	Moonlight	Security	High	07	Resolved
MLT-02	Moonlight	Security	High	08	Resolved
MLT-03	Moonlight	Logical	High	09	Resolved
MLT-04	Moonlight	Logical	High	10	Resolved
MLT-05	Moonlight	Logical	Medium	11	Resolved
MLT-06	Moonlight	Logical	Medium	12	Resolved
MLT-07	Moonlight	Logical	Informational	13	Resolved

PRISMA SHIELD

Overview

Findings Summary

ID	Section	Type	Severity	Page	Status
MLT-08	Moonlight	Logical	Informational	14	Resolved
RDM-01	MoonlightRedeem	Security	High	16	Resolved
RDM-02	MoonlightRedeem	Logical	High	17-18	Resolved
RDM-03	MoonlightRedeem	Logical	High	19	Resolved
RDM-04	MoonlightRedeem	Logical	High	20	Resolved
RDM-05	MoonlightRedeem	Logical	Informational	21	Resolved

Contract Addresses

Moonlight
https://arbiscan.io/address/0x0a1694716DE67c98f61942b2cAB7Df7FE659c87A#code

MoonlightRedeem
https://arbiscan.io/address/0x209c4e58ffd5dff54dc4283e23c17052c91bc749#code

Audit Findings

Moonlight

MLT-01 - Security High Severity

The Gnosis multisig address 0x70fDFC034f2AB7Ab8E279f1A30d4Af2905F8C06D can't be used directly to deploy the contract on the blockchain, and therefore is not the contract owner on deployment. The deploying non-multisig address will need to explicitly call transferOwnership post-deployment. If ownership is not transferred to the Gnosis multisig address, the non-multisig address will have unilateral control over the contract, and will be able to call all admin functions unilaterally.

Recommendation

Call _transferOwnership(0x70fDFC034f2AB7Ab8E279f1A30d4Af2905F8C06D); in the constructor.

Resolution

The team has implemented the recommendation.

Audit Findings

Moonlight

MLT-02 - Security High Severity

The burnFrom function allows anyone to burn anyone else's tokens.

Recommendation
Remove the custom burnFrom implementation and instead inherit from the OpenZeppelin ERC20Burnable contract.

Resolution
The team has implemented the recommendation.

Audit Findings

Moonlight

MLT-03 - Logical High Severity

The contract does not have the burn or burnFrom functions implemented, which are required by other contracts.

Recommendation
Inherit from the OpenZeppelin ERC20Burnable contract.

Resolution
The team has implemented the recommendation.

Audit Findings

Moonlight

MLT-04 - Logical High Severity

addToBlacklist should not be allowed to add _feeReceiver and the WrappedMoonLight contract address to _blacklist. And similarly, setFeeReceiver should not allow setting an address that is in _blacklist. If these happen, the other contracts would stop working as expected.

Recommendation

Add require statements to addToBlacklist to reject an address that is _feeReceiver or WrappedMoonLight, and add a require statement to setFeeReceiver to reject an address that is in _blacklist.

Resolution

The team has entirely removed _feeReceiver and the WrappedMoonLight contract.

Audit Findings

Moonlight

MLT-05 - Logical Medium Severity

The initial total supply is set to 3M tokens, but it is supposed to be 4M.

Recommendation

Set _initialSupply = 4000000 * 10**18;

Resolution

The team has implemented the recommendation.

Audit Findings

Moonlight

MLT-06 - Logical Medium Severity

The msg.sender in the constructor and _feeReceiver are not whitelisted, and so their token transfers will be taxed.

Recommendation

Add msg.sender and _feeReceiver to _whitelist in the constructor, and setFeeReceiver should be modified to set _whitelist to true for the new _feeReceiver (and optionally false for the old _feeReceiver). Moreover, removeFromWhitelist should not be allowed to remove _feeReceiver from _whitelist.

Resolution

The team has removed _feeReceiver, whitelisting, and the fee-on-transfer from the contract.

Audit Findings

Moonlight

MLT-07 - Logical Informational Severity

In _beforeTokenTransfer, there is no need to call super._beforeTokenTransfer(from, to, amount); which has an empty implementation.

Recommendation

Remove super._beforeTokenTransfer(from, to, amount); from _beforeTokenTransfer.

Resolution

The team has implemented the recommendation.

Audit Findings

Moonlight

MLT-08 - Logical Informational Severity

- addToBlacklist, removeFromBlacklist, burn, and burnFrom are not called in this contract, yet they are set to public visibility
- Warning: Unused function parameter. Remove or comment out the variable name to silence this warning: the amount parameter in _beforeTokenTransfer
- _beforeTokenTransfer does not modify any state variables, but its mutability is not set to view

Recommendation

- Change addToBlacklist, removeFromBlacklist, burn, and burnFrom from public to external
- Change _beforeTokenTransfer to:

```
function _beforeTokenTransfer(address from, address to, uint256) internal
view whenNotPaused override {
    ...
}
```

Resolution

The team has implemented the recommendations, and removed the custom implementation of burn and burnFrom.

Overview

Moonlight

- This contract is an ERC20 token that has added centralization of control. It allows the contract owner (Gnosis multisig address 0x70fDFC034f2AB7Ab8E279f1A30d4Af2905F8C06D) to blacklist addresses from transferring the token and also to disable token transfers in general.

- addToBlacklist and removeFromBlacklist can be used to add or remove an address from the token transfer blacklist. The contract owner has admin powers to call these functions.

- pause and unpause can be use to disable or enable token transfers. The contract owner has admin powers to call these functions.

Audit Findings

MoonlightRedeem

RDM-01 - Security High Severity

addToken allows adding already added tokens, which can be abused to redeem more tokens than allowed.

Recommendation
Modify addToken to not allow adding tokens that have already been added.

Resolution
The team has implemented the recommendation.

PRISMA SHIELD

Audit Findings

MoonlightRedeem

RDM-02 - Logical High Severity

Because of the redeem mechanism and the redeemFee, this contract runs the risk of potentially locking in tokens (like USDT) forever. For example, if the redeemFee is never changed to 0, or if no one owns any Moonlight tokens.

Recommendation

Add a way to extract tokens from this contract in such cases:

```
uint256 public lastRedeem;

constructor(address _moonlight) {
    ...
    lastRedeem = block.timestamp;
}

function redeem(uint256 amount) external {
    ...
    require(amount > 0, "Redeeming 0 is not allowed");
    lastRedeem = block.timestamp;
}

function recoverERC20(address tokenAddress, uint256 tokenAmount) external
onlyOwner {
    if(exists[tokenAddress]) {
        require(IERC20(moonlight).totalSupply() == 0 || block.timestamp - lastRedeem >
365 days, "Not allowed to recover");
    }
    IERC20(tokenAddress).transfer(IMoonlight(moonlight).owner(), tokenAmount);
}
```

Audit Findings

MoonlightRedeem

RDM-02 - Logical High Severity

Resolution

The team has implemented a recoverERC20 function which can be used by the contract owner to transfer any ERC20 token out of the contract. The function can be used at any time with no restrictions. The CrescentSwap team have explained that the MoonlightRedeem contract is funded by CrescentSwap + CrescentSwap's revenue, and therefore the contract owner reserves the right to deposit/withdraw from the MoonlightRedeem contract as they please.

Audit Findings

MoonlightRedeem

RDM-03 - Logical High Severity

The IMoonlight interface defines a getOwner function that is not implemented by the Moonlight contract.

Recommendation

Change getOwner to owner, in the IMoonlight interface and in the onlyOwner modifier definition.

Resolution

The team has implemented the recommendation.

Audit Findings

MoonlightRedeem

RDM-04 - Logical High Severity

getFloorPrice and amountToRedeem do not implement the correct decimals math, which will break when adding tokens to the contract with different decimals.

Recommendation

Change the code as follows:

```
function getFloorPrice(address token) external view returns (uint256) {
   return
      (IERC20(token).balanceOf(address(this)) *
         10 ** IERC20(moonlight).decimals()) /
      IERC20(moonlight).totalSupply();
}

function amountToRedeem(
   address token,
   uint256 amount
) public view returns (uint256) {
   return
      (IERC20(token).balanceOf(address(this)) * amount) /
      IERC20(moonlight).totalSupply();
}
```

Resolution

The team has implemented the recommendation.

Audit Findings

MoonlightRedeem

RDM-05 - Logical Informational Severity

The delete statement at the end of the redeem function does not have any benefit.

Recommendation

Remove the delete statement.

Resolution

The team has implemented the recommendation.

PRISMA SHIELD

Overview

MoonlightRedeem

- This contract burns Moonlight tokens from users in exchange for receiving tokens like USDT that have been added to the contract. The percentage of tokens received by users is the same as percentage of Moonlight tokens being burned of the totalSupply of Moonlight tokens, minus a redeemFee.

- The redeem function is used by users to burn the specified amount of Moonlight tokens, in exchange for receiving their percentage of the tokens in the MoonlightRedeem contract.

- The redeemFee publicly viewable variable contains the percentage deducted from the amount of tokens redeemed by users. It defaults to 2%, but can be changed to anything between 0% and 50%. The Moonlight owner has admin powers to change this value.

- The tokens publicly viewable variable contains the list of token addresses that can be redeemed by users. The Moonlight owner has admin powers to change this value.

- The amountToRedeemWithFee publicly viewable function can be used to return the amount of a token that will be received if the specified amount of Moonlight tokens is burned.

- The recoverERC20 function is used to remove any amount of any token from the contract. The Moonlight owner has admin powers to call this function.

How to Interpret Findings

Security - High Severity

Indicates that users' funds are at risk or that there is a high probability of exploitation.

Security - Medium Severity

No risk to the protocol or those who interact with it, however it should be highlighted nonetheless.

Logical - High Severity

Indicates that the errors puts users' funds at risk, or can result in significant functional failure in the code.

Logical - Medium Severity

Indicates some functional failure or discrepancy in the code.

Logical - Informational

Minor discrepancy between the intended functionality of the code and the implementation, which does not result in functional failure, or a recommendation to improve the logic.

Yellow Text

Indicates centralization of control and admin powers.

Red Text

An important warning to take note of.

Disclaimer

The information in this deep logic audit report objectively describes the smart contracts being audited, and points out logical and mathematical errors, security risks and vulnerabilities, and optimization opportunities in the audited code. This deep logic audit does not ensure the correctness or authenticity of any software or dApp that interacts with or claims to interact with any smart contract.

This audit report does not constitute any advice whatsoever. You are solely responsible for conducting your own due diligence and consulting your financial advisor before making any investment decisions. Trust in project owners is required to invest in this protocol as a Prisma Shield audit does not ensure the fulfillment of roadmap deliverables and allocation of funds. While our deep logic audits raise the level of security, reliability, mathematical accuracy, and logical soundness of the smart contracts reviewed, they do not amount to any form of warranty or guarantee that the reviewed smart contracts are void of any weaknesses, vulnerabilities, or bugs. Prisma Shield and its founders, employees, owners, and associates are not liable for any damage or loss of funds. Please ensure trust in the team prior to investing as this deep logic audit does not guarantee the promised use of your funds.

PRISMA SHIELD

Introducing Deep Logic Smart Contract Auditing to Web3

prismashield.com

prismashield@gmail.com

PrismaShield

PrismaShield