PRISMA SHIELD

CRESCENTSWAP

DEEP LOGIC AUDIT REPORT

CrescentSwap: Artemis Token and FeeRecipient Contracts

APR 06 2023

PRISMA SHIELD

Table of Contents

Overview

This audit only covers the Artemis and FeeRecipient contracts. It does not cover any other contracts built by CrescentSwap.

Please note that the FeeRecipient contract has a dependency on the GenesisArtemisApes smart contract (0x5dc5695cc991f277f47ecef73f5a016d8a938b94) which has not been audited by Prisma Shield. If said contract contains any errors, it could negatively affect the FeeRecipient contract and any user funds associated with it.

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	**12**	**8**	**4**
High Security Findings	2	0	2
Medium Security Findings	2	2	0
High Logical Findings	3	2	1
Medium Logical Findings	1	1	0
Informational Findings	4	3	1

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

Overview

Findings Summary

ID	Section	Type	Severity	Page	Status
ART-08	Artemis	Logical	Informational	14	Resolved
FEE-01	FeeRecipient	Security	High	17	Acknowledged
FEE-02	FeeRecipient	Security	High	18	Acknowledged
FEE-03	FeeRecipient	Logical	High	19	Acknowledged
FEE-04	FeeRecipient	Logical	High	20	Resolved

Contract Addresses

Artemis
https://arbiscan.io/address/0x3DD2E3005aE6Eda0D8A3967F6fC0799c4C842A08#code

FeeRecipient
https://arbiscan.io/address/0x87503DF392591D9eD31EFb0F265c795D489f8653#code

Audit Findings

Artemis

ART-01 - Logical High Severity

In tokensToMint, the math does not match the promise made in the documentation. It is presented in the code as:

_totalSupply * received / (totalBacking + 8% * received) * 92%

When it should simply be:

_totalSupply * received / totalBacking * 92%

Recommendation
Modify the code to be
_totalSupply.mul(received).div(totalBacking).mul(mintFee).div(feeDenominator);

Resolution
The team has implemented the recommendation.

Audit Findings

Artemis

ART-02 - Security Medium Severity

In addition to approve, the functions increaseAllowance and decreaseAllowance should be implemented to mitigate the well-known issues around setting allowance (https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729).

Recommendation

Add the increaseAllowance and decreaseAllowance implementations:

```
function increaseAllowance(address spender, uint256 addedValue) external returns (bool) {
    uint256 amount = _allowances[msg.sender][spender].add(addedValue);
    _allowances[msg.sender][spender] = amount ;
    emit Approval(msg.sender, spender, amount);
    return true;
}

function decreaseAllowance(address spender, uint256 subtractedValue) external returns (bool) {
    uint256 amount = _allowances[msg.sender][spender].sub(subtractedValue, "Decreased allowance below zero");
    _allowances[msg.sender][spender] = amount ;
    emit Approval(msg.sender, spender, amount);
    return true;
}
```

Resolution

The team has implemented the recommendation.

Audit Findings

Artemis

ART-03 - Security Medium Severity

setFees does not check that the fees are less than or equal to 10**5. WIthout this check, if the contract owner sets the fees to a number larger than 10**5, this can have adverse effects on user funds.

Recommendation

Add the check that the fees are less than are equal to 10**5 in setFees.

Resolution

The team has implemented the recommendation.

Audit Findings

Artemis

ART-04 - Logical Medium Severity

In transfer, if the recipient is the msg.sender, then this triggers _sell. This may result in unexpected behavior for a user who intends to just transfer the tokens to themselves (e.g. for testing or any other reason). Moreover, the same behavior should be implemented in transferFrom for consistency.

Recommendation

Remove the call to to _sell in transfer and just call _transferFrom, and change _transferFrom as such:

```
function transfer(address recipient, uint256 amount) external override nonReentrant
returns (bool) {
    return _transferFrom(msg.sender, recipient, amount);
}

function _transferFrom(address sender, address recipient, uint256 amount) internal
returns (bool) {
    require(recipient != address(0) && sender != address(0), "Transfer To Zero");
    if (sender == recipient) {
      require(_balances[sender] >= amount, "Insufficient Balance");
      emit Transfer(sender, recipient, amount);
      return true;
    }
    ...
}
```

Resolution

The team has implemented the recommendation.

Audit Findings

Artemis

ART-05 - Logical Informational Severity

In transfer and _transferFrom, if the sender == recipient, a Transfer event is emitted without checking that _balances[sender] is larger than or equal to the amount being transferred.

This is a "visual" bug, and does not have any side effects on the smart contract other than, if user A is approved to transfer from user B, and then user A transfers from user B to user B more than user B's balance, then this will change the approval even though the transaction is supposed to fail and revert.

Recommendation

Before the Transfer event is emitted, check that _balances[sender] is enough. See the implementation provided in ART-04.

Resolution

The team has acknowledged that this is a minor "visual" bug and has not implemented the fix.

Audit Findings

Artemis

ART-06 - Logical Informational Severity

setFeeRecipient does not set isTransferFeeExempt to false for the old feeRecipient.

Recommendation

Set isTransferFeeExempt[feeRecipient] = false; before updating the feeRecipient.

Resolution

The team has implemented the recommendation.

Audit Findings

Artemis

ART-07 - Logical Informational Severity

_mintWithBacking does not check that the recipient is not the zero address.

Recommendation

Add the check require(recipient != address(0), 'Zero Address'); to _mintWithBacking.

Resolution

The team has implemented the recommendation.

Audit Findings

Artemis

ART-08 - Logical Informational Severity

A couple of gas-saving opportunities:

- The checks in _mintWithBacking for the user's USDT balance are not required, because the USDT transfer function will fail anyways if the user's balance is insufficient
- There is no need for block.timestamp + 300 in the router.swapExactETHForTokens function call, simply using block.timestamp is sufficient

Recommendation

Remove the user USDT balance checks in _mintWithBacking, and just use block.timestamp in the router.swapExactETHForTokens function call.

Resolution

The team has implemented the recommendation.

Overview

Artemis

- This contract is an ERC20 token that is backed by USDT stored in the contract. The token can be bought and sold directly through the contract, but the amounts received are less than the backing price, which means that the backing price increases as the token is bought, sold, and transferred. If USDT loses its peg to the dollar, then the Artemis token will also lose its value.

- When the token is transferred from an address that is not isTransferFeeExempt to an address that is also not isTransferFeeExempt, the receiving address receives transferFee percentage (defaults to 92%) of the amount transferred. feeRecipientPercentage (defaults to 50%) of the remainder of the amount (4% of the amount transferred) is sent to the feeRecipient address, and anything that is left (final 4% of the amount transferred) is burned. The contract owner has admin powers to change isTransferFeeExempt and transferFee (minimum 90%), while the feeRecipientSetter has admin powers to change the feeRecipient and feeRecipientPercentage (up to 100%).

- mintWithNative and mintWithBacking can be used to mint new Artemis tokens (the former using ETH, and the latter using USDT). The address performing the mint, if not isTransferFeeExempt, will mint mintFee percentage (defaults to 92%) of the amount that is based on the backing price. feeRecipientPercentage (defaults to 50%) of the remainder of the amount based on the backing price (4%) will be minted to the feeRecipient address. The contract owner has admin powers to change the mintFee (minimum 90%), and can also disable/enable these functions to prevent/allow new token mints. They can also change the router used to swap ETH to USDT (defaults to UniswapV2Router).

- sell can be used to sell the Artemis token for USDT. If the seller is not isTransferFeeExempt, sellFee percentage (defaults to 92%) of the amount is sold to USDT. feeRecipientPercentage (defaults to 50%) of the remainder of the unsold tokens (4%) is sent to the feeRecipient address, and anything that is left (4%) is burned. This function cannot be disabled. The contract owner has admin powers to change the sellFee (minimum 90%).

Overview

Artemis

- The calculatePrice externally viewable variable returns the USDT backing price of 1 Artemis token.
- The amountOut externally viewable variable returns the USDT backing price of the specified number of Artemis tokens.
- The getValueOfHoldings externally viewable variable returns the USDT backing price of the Artemis holdings of the specified user address.
- The contract owner has admin powers to withdraw any token from the Artemis contract other than USDT.
- The contract owner has admin powers to change the contract owner as well as the feeRecipientSetter.

Audit Findings

FeeRecipient

FEE-01 - Security High Severity

This contract has a dependency on the 0x5Dc5695Cc991f277f47EcEF73f5A016d8a938B94 GenesisArtemisApes contract which is not being audited by Prisma Shield, and therefore code paths that rely on that contract might have logical errors or security vulnerabilities.

Recommendation

It is advisable that this contract is also audited to ensure correctness.

Resolution

The team declined to have that contract audited by Prisma Shield. Their reasoning was that it is a simple NFT contract that does not require an audit.

Audit Findings

FeeRecipient

FEE-02 - Security High Severity

The withdraw function allows only dev0 to call it and withdraw any token from the contract, including the Artemis token.

Recommendation

This function should be removed, or it should be modified to not allow withdrawing the Artemis token.

Resolution

The team confirmed that the withdraw function was added for safety net in case of extreme circumstance. The team also confirmed that dev0 is set to the address of a doxxed owner.

Audit Findings

FeeRecipient

FEE-03 - Logical High Severity

distributeBalance has a loop over an externally modifiable length, which could lead to out-of-gas errors if the loop grows too large, which would lock out the trigger function in this contract from being called.

Recommendation

Loops should be limited in size to avoid this class of errors.

Resolution

The team confirmed that the external variable is controlled via owner and not manipulatable unless under extreme circumstances.

Audit Findings

FeeRecipient

FEE-04 - Logical High Severity

distributeBalance returns if the totalSupply() >= 0, and therefore never distributes the funds to the NFT holders.

Recommendation
That if-condition should be changed to if (totalSupply() == 0).

Resolution
The team has implemented the recommendation.

Overview

FeeRecipient

- This contract receives Artemis tokens and distributes them, 1/4th of the tokens are split equally between 4 addresses: dev0, dev1, dev2, and dev3. The remaining 3/4th are split equally between NFT holders.

- The trigger function is what distributes the tokens as described above. Only one of dev0, dev1, dev2, or dev3 can call this function.

- The withdraw function can be used to withdraw any token from the contract, including the Artemis token (without having it be destributed to any of the other addresses). dev0 has admin powers to call this function.

- Each of dev0, dev1, dev2, and dev3 has the ability to change only their address.

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.

In its current form, the Artemis token solely relies on transaction volume as its driving mechanism. It is collateralized by the USDT token (0xFd086bC7CD5C481DCC9C85ebE478A1C0b69FCbb9) which has not been audited by Prisma Shield. If USDT encounters issues of any kind that cause loss of value, ARTMS will be affected.

PRISMA SHIELD

Introducing Deep Logic Smart Contract Auditing to Web3

prismashield.com

prismashield@gmail.com

PrismaShield

PrismaShield