Competitive Security Assessment

WestCoast NFT

Apr 27th, 2023

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

Overview

Project Detail

Project Name	WestCoast NFT
Platform & Language	Solidity
Codebase	• https://github.com/westcoastnft/mythos-contracts • audit commit - dcdcb3f8bb50ce98f66aa24c3a4d9229fcaed051 • final commit - 6c0aedb173522a07b8f66cbd99cf130ac2fb2292
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

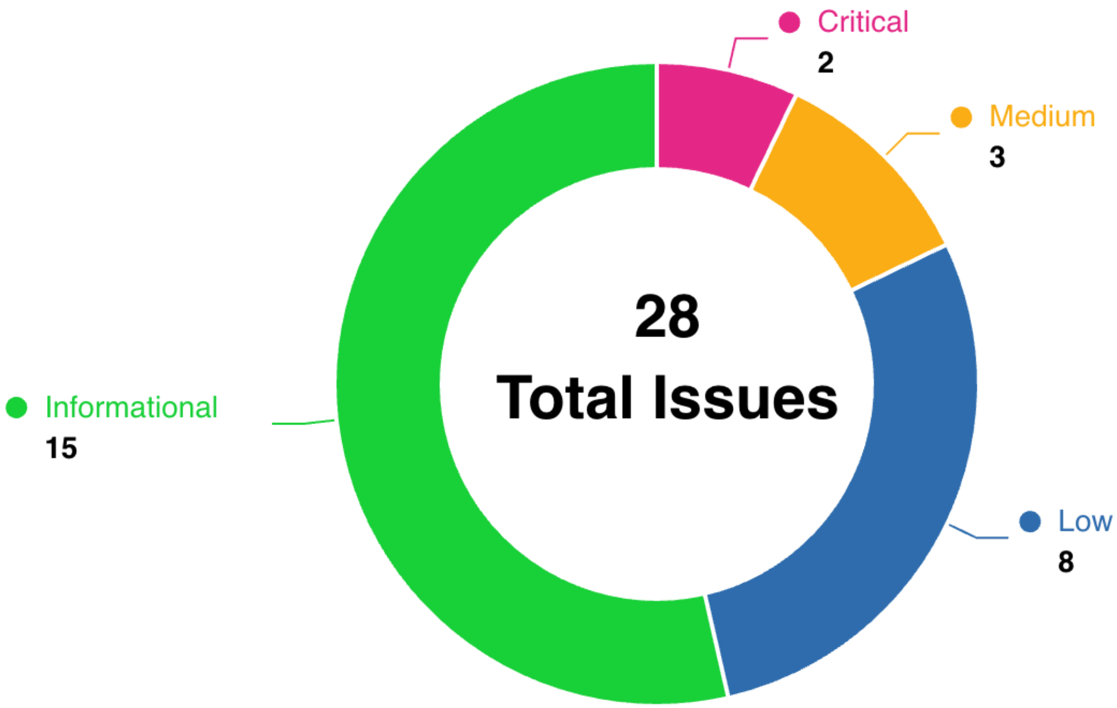
Code Vulnerability Review Summary

Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	2	0	0	1	0	1
Medium	3	0	3	0	0	0
Low	8	0	3	3	0	2
Informational	15	0	5	7	0	3

Audit Scope

File	Commit Hash
contracts/Mythos.sol	dcdcb3f8bb50ce98f66aa24c3a4d9229fcaed051
contracts/lib/SteppedDutchAuction.sol	dcdcb3f8bb50ce98f66aa24c3a4d9229fcaed051
contracts/lib/ERC721AOpensea.sol	dcdcb3f8bb50ce98f66aa24c3a4d9229fcaed051
contracts/external/ClosedSeaOperatorFilterer-1.0.0.sol	dcdcb3f8bb50ce98f66aa24c3a4d9229fcaed051
contracts/lib/WCNFTToken.sol	dcdcb3f8bb50ce98f66aa24c3a4d9229fcaed051
contracts/lib/IWCNFTErrorCodes.sol	dcdcb3f8bb50ce98f66aa24c3a4d9229fcaed051

Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
WCN-1	Address can be different in different chain	Logical	Medium	Acknowledged	parth_15
WCN-2	Don't use payable for user functions in `ERC721AOpensea` that don't handle the transfer	Code Style	Informational	Acknowledged	zaskoh
WCN-3	Gas Optimization - Cache array length outside of loop	Gas Optimization	Informational	Fixed	rajatbeladiya
WCN-4	Gas Optimization - Use calldata instead of memory for function arguments that do not get mutated	Gas Optimization	Informational	Fixed	rajatbeladiya

WCN-5	Gas Optimization: Using `external` market instead of `public` probably	Gas Optimization	Informational	Fixed	rajatbeladiya
WCN-6	Logical issue of `auctionTotal`	Logical	Low	Declined	Yaodao
WCN-7	Missing events on important state updates	Code Style	Informational	Acknowledged	Yaodao
WCN-8	NatSpec comments should be increased in contracts	Code Style	Informational	Fixed	parth_15
WCN-9	New Auction can be created if auction is paused	Logical	Critical	Fixed	parth_15, zaskoh
WCN-10	Potential divide by zero	Logical	Low	Fixed	Yaodao
WCN-11	Sending of tokens to the user can be done only once because `user.refundClaimed` cannot be set to `false`	Logical	Critical	Declined	parth_15, Yaodao
WCN-12	The owner can mint all of the NFTs	Privilege Related	Low	Declined	rajatbeladiya
WCN-13	There is no public burn function to burn the tokens	Logical	Low	Fixed	zaskoh, parth_15
WCN-14	Unlocked Pragma Version	Language Specific	Informational	Fixed	rajatbeladiya, parth_15, Yaodao
WCN-15	Unlocked Pragma Version	Language Specific	Informational	Fixed	zaskoh
WCN-16	Unnecessary centralisation risk in `Mythos` that can break NFT distribution	Privilege Related	Medium	Acknowledged	zaskoh, rajatbeladiya
WCN-17	Unused errors	Code Style	Informational	Declined	Yaodao
WCN-18	Unused internal function	Code Style	Informational	Fixed	Yaodao, rajatbeladiya
WCN-19	Using `constant` to save gas	Gas Optimization	Informational	Acknowledged	Yaodao
WCN-20	`WCNFTToken::setDefaultRoyalty` and `setTokenRoyalty` need more limit	Logical	Low	Acknowledged	rajatbeladiya, zaskoh

WCN-21	critical parameter should not change once auction is started	Logical	Medium	Acknowledged	rajatbeladiya
WCN-22	getRemainingSaleTime() returns wrong value if auction is not started	Logical	Informational	Declined	rajatbeladiya
WCN-23	malicious user can prevent the owner to refund funds to users	Logical	Low	Fixed	rajatbeladiya
WCN-24	pausing the auction will skip the auction's time duration	Logical	Low	Acknowledged	rajatbeladiya
WCN-25	preBuyActive can't be set to same value	Logical	Informational	Declined	parth_15
WCN-26	provenance is not used in the contract	Gas Optimization	Informational	Acknowledged	parth_15
WCN-27	setMinimumPreBuyPrice lacks sanity check	Logical	Low	Acknowledged	parth_15
WCN-28	users may lose bid amount	Logical	Informational	Acknowledged	rajatbeladiya

WCN-1:Address can be different in different chain

Category	Severity	Code Reference	Status	Contributor
Logical	**Medium**	• **code/contracts/lib/ERC721AOpensea.sol#L97**	**Acknowledged**	**parth_15**

Code

```
97:         return operator == address(0x1E0049783F008A0085193E00003D00cd54003c71);
```

Description

parth_15 : Add a setAddress function to set different address for different chain instead of hardcoding.

Recommendation

parth_15 : Use different address for different chain instead of hardcoding.

Client Response

- We have copied the implementation from here https://github.com/Vectorized/closedsea/blob/0ae1c8d267cac0b05e87106835880b6e2be77631/src/example/ExampleERC7
- We have left the code as is. Mostly because opensea deploys to deterministic address using create2, so this exists at the same address in both goerli and mainnet.

WCN-2:Don't use payable for user functions in `ERC721AOpensea` that don't handle the transfer

Category	Severity	Code Reference	Status	Contributor
Code Style	**Informational**	• **code/contracts/lib/ERC721AOpensea.sol#L32-L36** • **code/contracts/lib/ERC721AOpensea.sol#L45** • **code/contracts/lib/ERC721AOpensea.sol#L53** • **code/contracts/lib/ERC721AOpensea.sol#L62**	**Acknowledged**	**zaskoh**

Code

```
32:    function approve(address operator, uint256 tokenId)
33:        public
34:        payable
35:        override(IERC721A, ERC721A)
36:        onlyAllowedOperatorApproval(operator)

45:    ) public payable override(IERC721A, ERC721A) onlyAllowedOperator(from) {

53:    ) public payable override(IERC721A, ERC721A) onlyAllowedOperator(from) {

62:    ) public payable override(IERC721A, ERC721A) onlyAllowedOperator(from) {
```

Description

zaskoh : Currently the functions `approve`, `transferFrom` and both `safeTransferFrom` use payable in `ERC721AOpensea`. All of the functions don't expect to receive native tokens and don't track the values a user sent here.
You should only use payable on functions that expect to receive native tokens. Tokens sent by accident by a user are lost, or at least it depends on the admin to recover it for the user.

Recommendation

zaskoh : Remove `payable` from the functions. You will need to overwrite the `ERC721A.sol` implementation or as an alternative you could consider using the openzeppelin implementations for ERC721 or solmate for a more gas efficient implementation.

Client Response

- We realize they have added a payable modifier in ERC721A 4.2.2 to 4.2.3 . The operator filter registry provided by churi-labs also targets 4.2.3 and has some syntax we did not go through; hence, we copied their examples and targeted the same versions.

WCN-3:Gas Optimization - Cache array length outside of loop

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/contracts/Mythos.sol#L356-L358	Fixed	rajatbeladiya

Code

```
356:        for (uint256 i; i < receivers.length; i++) {
357:            _sendTokensAndRefund(receivers[i]);
358:        }
```

Description

rajatbeladiya : Cache array length outside of loop in `sendTokensAndRefundBatch()` function. If not cached, the solidity compiler will always read the length of the array during each iteration. That is, if it is a storage array, this is an extra sload operation (100 additional extra gas for each iteration except for the first) and if it is a memory array, this is an extra mload operation (3 additional gas for each iteration except for the first).

Recommendation

rajatbeladiya : Consider below fix in the `mythos.sendTokensAndRefundBatch()` function

```
uint256 receiversLength = receivers.length;
for (uint256 i; i < receiversLength; i++) {
   _sendTokensAndRefund(receivers[i]);
}
```

Client Response

- We have implemented the recommendation provided

WCN-4:Gas Optimization - Use calldata instead of memory for function arguments that do not get mutated

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/contracts/Mythos.sol#L356 • code/contracts/Mythos.sol#L404	Fixed	rajatbeladiya

Code

```
356:        for (uint256 i; i < receivers.length; i++) {

404:    function setProvenance(string memory provenance_)
```

Description

rajatbeladiya : Mark data types as `calldata` instead of `memory` where possible. This makes it so that the data is not automatically loaded into memory. If the data passed into the function does not need to be changed (like updating values in an array), it can be passed in as `calldata`. The one exception to this is if the argument must later be passed into another function that takes an argument that specifies `memory` storage.

Recommendation

rajatbeladiya : change data types as `calldata` instead of `memory`

Client Response

- We have implemented the recommendation provided

WCN-5:Gas Optimization: Using `external` market instead of `public` probably

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/contracts/lib/SteppedDutchAuction.sol#L163 • code/contracts/Mythos.sol#L279	Fixed	rajatbeladiya

Code

```
163:    function getAuctionPrice() public view returns (uint256) {

279:    function amountPurchased(address a) public view returns (uint256) {
```

Description

rajatbeladiya : Functions visibility that is not used internally could be marked as external

Recommendation

rajatbeladiya : change functions visibility to external

Client Response

- We have changed `amountPurchased` to be external; however, we left getAuctionPrice as public because it is being used in the main Mythos contract.

WCN-6:Logical issue of `auctionTotal`

Category	Severity	Code Reference	Status	Contributor
Logical	Low	code/contracts/Mythos.sol#L33 code/contracts/Mythos.sol#L154-L175 code/contracts/Mythos.sol#L226-L250 code/contracts/Mythos.sol#L300-L328	Declined	Yaodao

Code

```
33:    uint256 public auctionTotal;

154:    function deposit() external payable isPreBuyActive {
155:        User storage bidder = userData[msg.sender]; // get user's current bid total
156:        uint256 contribution_ = bidder.contribution; // bidder.contribution is uint216
157:        uint256 auctionTotal_ = auctionTotal;
158:        unchecked {
159:            // does not overflow
160:            contribution_ += msg.value;
161:            auctionTotal_ += msg.value;
162:            auctionTotal = auctionTotal_;
163:        }
164:
165:        if (contribution_ < minimumPreBuyPrice)
166:            revert LowerThanMinimumDepositAmount();
167:        bidder.contribution = uint216(contribution_);
168:        emit Deposit(
169:            msg.sender,
170:            msg.value,
171:            contribution_,
172:            auctionTotal_,
173:            minimumPreBuyPrice
174:        );
175:    }

226:    function bid() external payable isAuctionActive {
227:        User storage bidder = userData[msg.sender]; // get user's current bid total
228:        uint256 contribution_ = bidder.contribution; // bidder.contribution is uint216
229:        uint256 auctionTotal_ = auctionTotal;
230:        unchecked {
231:            // does not overflow
232:            numberOfBids++;
233:            contribution_ += msg.value;
234:            auctionTotal_ += msg.value;
235:            auctionTotal = auctionTotal_;
236:        }
237:
238:        uint256 currentAuctionPrice = getAuctionPrice();
239:        if (contribution_ < currentAuctionPrice)
240:            revert LowerThanMinimumBidAmount();
```

```
        bidder.contribution = uint216(contribution_);
        emit Bid(
            msg.sender,
            msg.value,
            contribution_,
            auctionTotal_,
            numberOfBids,
            currentAuctionPrice
        );
    }

    function _sendTokensAndRefund(ReceiverData calldata receiver) internal {
        uint256 price_ = price;

        address to = receiver.to;
        uint32 numberOfTokens = receiver.numberOfTokens;
        User storage user = userData[to]; // get user data
        uint256 userContribution = user.contribution;

        // send tokens first, calculate the maximum amount of tokens purchased
        uint256 maxNumberOfTokens = _amountPurchased(userContribution, price_);
        if (numberOfTokens > maxNumberOfTokens)
            revert SendingMoreThanPurchased();

        if (numberOfTokens > 0) {
            if (user.tokensClaimed != 0) revert TokensAlreadySent();
            user.tokensClaimed = uint32(numberOfTokens);
            _internalMint(to, numberOfTokens);
        }

        // send refund
        if (user.refundClaimed) revert RefundClaimed();
        user.refundClaimed = true;
        uint256 refundValue = userContribution - (price_ * numberOfTokens);

        if (refundValue > 0) {
            (bool success, ) = to.call{value: refundValue}("");
            if (!success) revert RefundFailed(to);
        }
    }
```

Description

Yaodao : - WCNFT needs `auctionTotal` for offchain calculations.

- The auction will only be run once.
- It makes sense to maintain `auctionTotal` at its value as of the end of bidding in case it needs to be re-checked during off chain calculations.

Recommendation

Yaodao : Recommend confirming the usage of the variable `auctionTotal` and adding the logic to reduce the refund amount.

Consider below fix in the `Mythos._sendTokensAndRefund` function

```
        if (refundValue > 0) {
            auctionTotal -= refundValue;
            (bool success, ) = to.call{value: refundValue}("");
            if (!success) revert RefundFailed(to);
        }
```

Client Response

- The auction will only be run once.
- It makes sense to maintain `auctionTotal` at its value as of the end of bidding in case it needs to be re-checked during off chain calculations.

WCN-7:Missing events on important state updates

Category	Severity	Code Reference	Status	Contributor
Code Style	Informational	• code/contracts/Mythos.sol#L131-L135 • code/contracts/Mythos.sol#L142-L147 • code/contracts/Mythos.sol#L184-L193 • code/contracts/Mythos.sol#L257-L260 • code/contracts/Mythos.sol#L300-L328 • code/contracts/Mythos.sol#L386-L391 • code/contracts/Mythos.sol#L404-L409	Acknowledged	Yaodao

Code

```
    function setPreBuyActive(bool state) external onlyRole(SUPPORT_ROLE) {
        if (auctionActive) revert DutchAuctionIsActive();
        if (price != 0) revert PriceHasBeenSet();
        preBuyActive = state;
    }

    function setMinimumPreBuyPrice(uint256 minimumPreBuyPriceInWei)
        external
        onlyRole(SUPPORT_ROLE)
    {
        minimumPreBuyPrice = minimumPreBuyPriceInWei;
    }

    function createNewAuction(
        uint256 startPrice,
        uint256 finalPrice,
        uint256 priceStep,
        uint256 timeStepSeconds
    ) external onlyRole(SUPPORT_ROLE) {
        if (auctionActive) revert DutchAuctionIsActive();

        _createNewAuction(startPrice, finalPrice, priceStep, timeStepSeconds);
    }

    function setPrice(uint256 priceInWei_) external onlyOwner {
        if (auctionActive) revert UserCanStillAddBids();
        price = priceInWei_;
    }

    function _sendTokensAndRefund(ReceiverData calldata receiver) internal {
        uint256 price_ = price;

        address to = receiver.to;
        uint32 numberOfTokens = receiver.numberOfTokens;
        User storage user = userData[to]; // get user data
        uint256 userContribution = user.contribution;

        // send tokens first, calculate the maximum amount of tokens purchased
        uint256 maxNumberOfTokens = _amountPurchased(userContribution, price_);
        if (numberOfTokens > maxNumberOfTokens)
            revert SendingMoreThanPurchased();

        if (numberOfTokens > 0) {
            if (user.tokensClaimed != 0) revert TokensAlreadySent();
            user.tokensClaimed = uint32(numberOfTokens);
            _internalMint(to, numberOfTokens);
        }

        // send refund
        if (user.refundClaimed) revert RefundClaimed();
        user.refundClaimed = true;
        uint256 refundValue = userContribution - (price_ * numberOfTokens);

        if (refundValue > 0) {
            (bool success, ) = to.call{value: refundValue}("");
            if (!success) revert RefundFailed(to);
        }
    }

    function setBaseURI(string memory baseURI_)
        external
        onlyRole(SUPPORT_ROLE)
    {
        _baseURIextended = baseURI_;
    }

    function setProvenance(string memory provenance_)
        external
        onlyRole(SUPPORT_ROLE)
    {
        provenance = provenance_;
    }
```

Description

Yaodao : Functions that update state variables should emit relevant events as notifications.

Recommendation

Yaodao : Recommend adding events for state-changing actions, and emitting them in their relevant functions.

Client Response

- Most state changes referred to (`setBaseURI()` , `setProvenance()` , etc) are called once, and can easily be confirmed by calling the associated `view` functions.
- Event emission during `sendTokensAndRefund()` would considerably increase gas cost, and add very little useful info (if a user receives a token or refund, there will be an associated tx for it).

WCN-8:NatSpec comments should be increased in contracts

Category	Severity	Code Reference	Status	Contributor
Code Style	Informational	• code/contracts/lib/ERC721AOpensea.sol#L8-L99	Fixed	parth_15

Code

```
abstract contract ERC721AOpensea is
    ERC721AQueryable,
    OperatorFilterer,
    Ownable
{
    bool public operatorFilteringEnabled;

    constructor() {
        _registerForOperatorFiltering();
        operatorFilteringEnabled = true;
    }

    /***************************************************************************
     * Operator Filterer
     */

    function setApprovalForAll(address operator, bool approved)
        public
        override(IERC721A, ERC721A)
        onlyAllowedOperatorApproval(operator)
    {
        super.setApprovalForAll(operator, approved);
    }

    function approve(address operator, uint256 tokenId)
        public
        payable
        override(IERC721A, ERC721A)
        onlyAllowedOperatorApproval(operator)
    {
        super.approve(operator, tokenId);
    }

    function transferFrom(
        address from,
        address to,
        uint256 tokenId
    ) public payable override(IERC721A, ERC721A) onlyAllowedOperator(from) {
        super.transferFrom(from, to, tokenId);
    }

    function safeTransferFrom(
        address from,
        address to,
        uint256 tokenId
    ) public payable override(IERC721A, ERC721A) onlyAllowedOperator(from) {
        super.safeTransferFrom(from, to, tokenId);
    }

    function safeTransferFrom(
        address from,
        address to,
        uint256 tokenId,
        bytes memory data
    ) public payable override(IERC721A, ERC721A) onlyAllowedOperator(from) {
        super.safeTransferFrom(from, to, tokenId, data);
    }

    function supportsInterface(bytes4 interfaceId)
        public
        view
        virtual
        override(IERC721A, ERC721A)
        returns (bool)
    {
        // Supports the following `interfaceId`s:
        // - IERC165: 0x01ffc9a7
        // - IERC721: 0x80ac58cd
        // - IERC721Metadata: 0x5b5e139f
        return ERC721A.supportsInterface(interfaceId);
    }

    function setOperatorFilteringEnabled(bool value) public onlyOwner {
        operatorFilteringEnabled = value;
    }

    function _operatorFilteringEnabled() internal view override returns (bool) {
        return operatorFilteringEnabled;
    }

    function _isPriorityOperator(address operator)
        internal
        pure
        override
        returns (bool)
    {
        // OpenSea Seaport Conduit:
        // https://etherscan.io/address/0x1E0049783F008A0085193E00003D00cd54003c71
        // https://goerli.etherscan.io/address/0x1E0049783F008A0085193E00003D00cd54003c71
        return operator == address(0x1E0049783F008A0085193E00003D00cd54003c71);
    }
}
```

Description

parth_15 : It is recommended that Solidity contracts are fully annotated using NatSpec for all public interfaces (everything in the ABI). It is clearly stated in the Solidity official documentation. In complex projects such as Defi, the interpretation of all functions and their arguments and returns is important for code readability and auditability. https://docs.soliditylang.org/en/v0.8.15/natspec-format.html

Recommendation

parth_15 : NatSpec comments should be increased in Contracts

Consider below fix in the `sample.test()` function

```
/**
     * @notice Sets approval for specific withdrawer addresses
     * @dev This function updates the amount of the withdrawApproval mapping value based on the give
n 3 argument values
     * @param withdrawer_ Withdrawer address from state variable
     * @param token_ instance of ERC20
     * @param amount_ User amount
     * @param permissioned Control of admin for access
     */
    function setApprovalFor(
        address withdrawer_,
        ERC20 token_,
        uint256 amount_
    ) external permissioned {
        withdrawApproval[withdrawer_][token_] = amount_;
        emit ApprovedForWithdrawal(withdrawer_, token_, amount_);
    }
```

Client Response

- We have implemented comments for the ERC721AOpensea contract

WCN-9:New Auction can be created if auction is paused

Category	Severity	Code Reference	Status	Contributor
Logical	Critical	• code/contracts/Mythos.sol#L142 • code/contracts/Mythos.sol#L184 • code/contracts/Mythos.sol#L199 • code/contracts/Mythos.sol#L217 • code/contracts/Mythos.sol#L257 • code/contracts/Mythos.sol#L404	Fixed	parth_15, zaskoh

Code

```
142:    function setMinimumPreBuyPrice(uint256 minimumPreBuyPriceInWei)

184:    function createNewAuction(

199:    function startAuction() external onlyRole(SUPPORT_ROLE) {

217:    function endAuction() external onlyRole(SUPPORT_ROLE) {

257:    function setPrice(uint256 priceInWei_) external onlyOwner {

404:    function setProvenance(string memory provenance_)
```

Description

parth_15 : `SUPPORT_ROLE` can start new auction by calling `createNewAuction`. If auction is not active, it is possible to create a new auction. It is possible that the previous auction is paused for a while and due to that, `auctionActive` will be `false`. In this case, new auction can still be created even if users had placed bids and deposited in previous auction.

The impact is that user's bids and deposits to previous auction is invalid and new auction will overwrite the values of previous auction. The user will lose the trust from the protocols if multiple auctions are created.

zaskoh : The following functions should only be called once to ensure trustlessness and integrity of the dutch auction and the NFT contract and the mint process in general.

- `setProvenance` is used as a proofe of a fair drop of the NFT before the tokens are revealed, if updated later on can break this assumption
- `setMinimumPreBuyPrice` if updated while pre buy is active can manipulate the price for users and move early users in a possible bad position

- `createNewAuction` this will reset the dutch auction and can be used to manipulate the price
- `startAuction` the dutch auction should only be started once
- `setPrice` the price is used in the current implementation to distribute the mint and should not be updated

Updating or chaning the values in this functions will break assumptions and trust from users.

Recommendation

parth_15 : Place a check that multiple auctions can not be created.Also if the developer want to pause the auction, they should create a function named `pauseAuction()` rather than use `endAuction()` because it will lead to misunderstanding.
zaskoh : To each of these setter functions add a check for (variable == 0) to ensure they are set once in a permanent way.
Example:

```
+    bool provenanceIsSet;
     function setProvenance(string memory provenance_)
         external
         onlyRole(SUPPORT_ROLE)
     {
+        require(!provenanceIsSet);
         provenance = provenance_;
+        provenanceIsSet = true;
     }
```

Client Response

- We have purposely left this as is since it may be possible to enter incorrect auction information, and should be able to adjust it before the auction starts
- We have altered the revert statement in the `createNewAuction`, such that it may not be modified after the dutch auction has started

```
function createNewAuction(...) ... {
        // replace this:
        if (auctionActive) revert DutchAuctionIsActive();

        // with this:
        if (startTime > 0) revert DutchAuctionHasStarted();

        // ...
    }
```

WCN-10:Potential divide by zero

Category	Severity	Code Reference	Status	Contributor
Logical	**Low**	• **code/contracts/Mythos.sol#L279**	**Fixed**	**Yaodao**

Code

```
279:    function amountPurchased(address a) public view returns (uint256) {
```

Description

Yaodao : The variable `price` is set by the owner and can be zero. The following function `amountPurchased()` is a public view function, so it can be called by the users. According to the current codes, the call will revert without corresponding error information when the price is 0.

Consider below function

```
    function amountPurchased(address a) public view returns (uint256) {
        return userData[a].contribution / price;
    }
```

Recommendation

Yaodao : Recommend adding check to give the corresponding error information.

Consider below fix in the function

```
    function amountPurchased(address a) public view returns (uint256) {
        if (price == 0) revert PriceHasNotBeenSet();
        return userData[a].contribution / price;
    }
```

Client Response

- We have implemented the recommendation provided

WCN-11:Sending of tokens to the user can be done only once because `user.refundClaimed` cannot be set to `false`

Category	Severity	Code Reference	Status	Contributor
Logical	Critical	• code/contracts/Mythos.sol#L226 • code/contracts/Mythos.sol#L314 • code/contracts/Mythos.sol#L320-L322	Declined	parth_15, Yaodao

Code

```
226:    function bid() external payable isAuctionActive {

314:            if (user.tokensClaimed != 0) revert TokensAlreadySent();

320:        if (user.refundClaimed) revert RefundClaimed();
321:        user.refundClaimed = true;
322:        uint256 refundValue = userContribution - (price_ * numberOfTokens);
```

Description

parth_15 : Consider a scenario where auction is active, user placed the bid and tokens are sent to them for their contribution and remaining proportion is refunded to them. Now, if user bids again for making contribution, their new funds contributed will be locked in the protocol and they won't be able to claim. This is because of the check present in `_sendTokensAndRefund`. It checks that user can't claim refund again if they claimed it once. So, for a contribution made second time by the user, they won't receive tokens for that and their invested funds are lost.

The impact is that the users can lose their funds if they try to `bid` once they receive token and refund earlier.

Consider following steps to reproduce the error:

- Auction is active
- User calls `bid` and sent some contribution
- Admin sent tokens to the user and remaining is refunded to the user. Thus, `user.tokensClaimed` will be `true` for that user.
- User against bids the amount.
- The new amount that user bid will be lost.

  Yaodao : According to the following codes, once the user refund, the `user.refundClaimed` will be marked as true and the user can never refund again because the user's information will never be reset.

```
    function _sendTokensAndRefund(ReceiverData calldata receiver) internal {
        ...
        // send refund
        if (user.refundClaimed) revert RefundClaimed();
        user.refundClaimed = true;
        uint256 refundValue = userContribution - (price_ * numberOfTokens);

        if (refundValue > 0) {
            (bool success, ) = to.call{value: refundValue}("");
            if (!success) revert RefundFailed(to);
        }
    }
```

In the protocol, the `SUPPORT_ROLE` can create a new auction after the current auction is over. Besides, the user can deposit and bid again because the `user.refundClaimed` is not checked in the functions `deposit()` and `bid()`. However, the call of `_sendTokensAndRefund()` will fail as the `user.refundClaimed` has been marked as true. As a result, the user's ETH will be locked in the contract and the user can't get back unless the owner withdraws and transfers to the user directly.

Recommendation

parth_15 : Impose a strict check to ensure that user can't bid again once they have got their tokens.
Yaodao : Recommend resetting the user's information after the refund in the function `_sendTokensAndRefund()`.

Client Response

- This is expected behaviour; the sendTokensAndRefund happens only at the end of the auction/pre buy phase

WCN-12:The owner can mint all of the NFTs

Category	Severity	Code Reference	Status	Contributor
Privilege Related	**Low**	• **code/contracts/Mythos.sol#L370-L376**	**Declined**	**rajatbeladiya**

Code

```
370:    function devMint(address to, uint256 numberOfTokens)
371:        external
372:        onlyRole(SUPPORT_ROLE)
373:        nonReentrant
374:    {
375:        _internalMint(to, numberOfTokens);
376:    }
```

Description

rajatbeladiya : In `Mythos.devMint()` function can mint an unrestricted amount of NFTs. There can be the scenario that the owner's private key is compromised and the attacker mint all of the NFTs during launch (Private keys are getting compromised very often in this space).

Recommendation

rajatbeladiya : Limit the number of NFTs the owner can mint. This will help with the trust of the protocol because buyers will know exactly how many NFTs can the Dev Team mint for themselves.

Client Response

- this is expected behaviour
- Based on previous discussions, it adds necessary flexibility for `SUPPORT_ROLE` in multiple circumstances.

WCN-13:There is no public burn function to burn the tokens

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/contracts/Mythos.sol#L434-L441 • code/contracts/Mythos.sol#L435-L436	Fixed	zaskoh, parth_15

Code

```
434:    function _burn(uint256 tokenId, bool approvalCheck)
435:        internal
436:        virtual
437:        override
438:    {
439:        super._burn(tokenId, approvalCheck);
440:        _resetTokenRoyalty(tokenId);
441:    }

435:        internal
436:        virtual
```

Description

zaskoh : Currently `Mythos` overrides the internal `_burn` function where the royalty is reset.

```
File: code/contracts/Mythos.sol
434:     function _burn(uint256 tokenId, bool approvalCheck)
435:         internal
436:         virtual
437:         override
438:     {
439:         super._burn(tokenId, approvalCheck);
440:         _resetTokenRoyalty(tokenId);
441:     }
```

Still in the current implementation it misses the possibility to `burn` a NFT.

parth_15 : The contract has internal `_burn` function that overrides `ERC721A` burn functionality and additionally add functionality to clear the royalty information for the token. However, there is no public `burn` function which can be called and thus this function is not usable.

The impact is that the users won't be able to burn their tokens and contract lacks one of it's intended functionality.

Recommendation

zaskoh : Implement a external function that a user can use to burn a NFT.

Example:

```
+    function burn(uint256 tokenId) external {
+        _burn(tokenId, true);
+    }
```

parth_15 : Add a public `burn` function which calls the internal `_burn` function.

Client Response

- We will remove the function. This was left in case a burn function was implemented, which would require a reset of token royalty

WCN-14:Unlocked Pragma Version

Category	Severity	Code Reference	Status	Contributor
Language Specific	**Informational**	• **code/contracts/lib/WCNFTToken.sol#L2** • **code/contracts/lib/SteppedDutchAuction.sol#L2** • **code/contracts/lib/IWCNFTErrorCodes.sol#L2** • **code/contracts/lib/ERC721AOpensea.sol#L2** • **code/contracts/external/ClosedSeaOperatorFilterer-1.0.0.sol#L2** • **code/contracts/Mythos.sol#L3**	**Fixed**	**rajatbeladiya, parth_15, Yaodao**

Code

```
2:pragma solidity ^0.8.4;

2:pragma solidity ^0.8.12;

2:pragma solidity ^0.8.12;

2:pragma solidity ^0.8.12;

2:pragma solidity ^0.8.12;

3:pragma solidity ^0.8.12;
```

Description

rajatbeladiya : WestCoastNFT solidity files have a pragma solidity version number with ^0.8.12. The caret (^) points to unlocked pragma, meaning the compiler will use the specified version or above.

parth_15 : Avoid floating pragmas for non-library contracts.

While floating pragmas make sense for libraries to allow them to be included with multiple different versions of applications, it may be a security risk for application implementations.

A known vulnerable compiler version may accidentally be selected or security tools might fall-back to an older compiler version ending up checking a different EVM compilation that is ultimately deployed on the blockchain.

Yaodao : The contracts cited have an unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to ambiguity when debugging, as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

rajatbeladiya : It's good practice to use specific solidity versions to know compiler bug fixes and optimisations were enabled at the time of compiling the contract.
parth_15 : It is recommended to pin to a concrete compiler version. For ex: pragma solidity ^0.8.12 can be changed to pragma.solidity 0.8.12
Yaodao : Recommend the compiler version is instead locked at the lowest version possible that the contract can be compiled at.

Client Response

- we have changed the pragma versions to `pragma solidity 0.8.13;`
- we have also changed the IWCNFT interface file pragma to be floating `^0.8.0`

WCN-15:Unlocked Pragma Version

Category	Severity	Code Reference	Status	Contributor
Language Specific	**Informational**	• **code/contracts/external/ClosedSeaOperatorFilterer-1.0.0.sol#L2** • **code/contracts/lib/ERC721AOpensea.sol#L2** • **code/contracts/lib/IWCNFTErrorCodes.sol#L2** • **code/contracts/lib/SteppedDutchAuction.sol#L2** • **code/contracts/lib/WCNFTToken.sol#L2** • **code/contracts/Mythos.sol#L3**	**Fixed**	**zaskoh**

Code

```
2:pragma solidity ^0.8.4;

2:pragma solidity ^0.8.12;

2:pragma solidity ^0.8.12;

2:pragma solidity ^0.8.12;

2:pragma solidity ^0.8.12;

3:pragma solidity ^0.8.12;
```

Description

zaskoh : For implementation contracts it's best to choose a fixed version like 0.8.17 (best to choose the version you have tested it against).
For your Interfaces it's best to choose a floating version like 0.8.0^ so other projects can import your original interfaces and dont need to rewrite or change them if they use a lower version than 0.8.X^.

Recommendation

zaskoh : Use a fixed version like 0.8.17 for your implementation contracts and 0.8.0^ for the interface IWCNFTErrorCodes.sol

Client Response

- we have changed the pragma versions to `pragma solidity 0.8.13;`
- we have also changed the IWCNFT interface file pragma to be floating `^0.8.0`

WCN-16:Unnecessary centralisation risk in `Mythos` that can break NFT distribution

Category	Severity	Code Reference	Status	Contributor
Privilege Related	Medium	- code/contracts/Mythos.sol#L300-L328 - code/contracts/Mythos.sol#L450-L455	Acknowledged	zaskoh, rajatbeladiya

Code

```
    function _sendTokensAndRefund(ReceiverData calldata receiver) internal {
        uint256 price_ = price;

        address to = receiver.to;
        uint32 numberOfTokens = receiver.numberOfTokens;
        User storage user = userData[to]; // get user data
        uint256 userContribution = user.contribution;

        // send tokens first, calculate the maximum amount of tokens purchased
        uint256 maxNumberOfTokens = _amountPurchased(userContribution, price_);
        if (numberOfTokens > maxNumberOfTokens)
            revert SendingMoreThanPurchased();

        if (numberOfTokens > 0) {
            if (user.tokensClaimed != 0) revert TokensAlreadySent();
            user.tokensClaimed = uint32(numberOfTokens);
            _internalMint(to, numberOfTokens);
        }

        // send refund
        if (user.refundClaimed) revert RefundClaimed();
        user.refundClaimed = true;
        uint256 refundValue = userContribution - (price_ * numberOfTokens);

        if (refundValue > 0) {
            (bool success, ) = to.call{value: refundValue}("");
            if (!success) revert RefundFailed(to);
        }
    }

    function withdraw() external onlyOwner nonReentrant {
        (bool success, ) = shareholderAddress.call{
            value: address(this).balance
        }("");
        if (!success) revert WithdrawFailed();
    }
```

Description

zaskoh : Currently it's possibile for the owner to call `Mythos.withdraw` and move all tokens to the associated `shareholderAddress`.

If this function is called before all the NFT's are distributed via `sendTokensAndRefund` / `sendTokensAndRefundBatch`, it will break the contract, as the `_sendTokensAndRefund` function will not work in case he should get a refund.

To prevent users from such a rugpull, it's better to track what the owner is allowed to withdraw and only allow him to withdraw that amount.

rajatbeladiya : the contract owner can withdraw contract funds by calling `withdraw()` function. There is no payable `receive()` function to replenish the contract again for the refund process. So if the contract owner withdraws more than it should be allowed to, there will be not enough funds to refund process.

Recommendation

zaskoh : You should implement a way to track how much the owner is allowed to withdraw.

Example:

```
         user.refundClaimed = true;
         uint256 refundValue = userContribution - (price_ * numberOfTokens);
+        maximumWithdraw = price_ * numberOfTokens; // @audit add in _sendTokensAndRefund to track how much admin is allowed to withdraw

         if (refundValue > 0) {

-    function withdraw() external onlyOwner nonReentrant {
+    uint256 maximumWithdraw;
+    function withdraw() external onlyOwner nonReentrant {
+        uint256 withdrawAmount = maximumWithdraw;
+        maximumWithdraw = 0;
         (bool success, ) = shareholderAddress.call{
-            value: address(this).balance
+            value: withdrawAmount
         }("");
         if (!success) revert WithdrawFailed();
     }
```

rajatbeladiya : the owner should be allowed to withdraw after the refund process is completed.

Client Response

- wcnft is aware of this risk (discussed in bucket auction)
- consensus was that:
    - the client is established and trusted,

    - the client is aware that strict procedure should be followed for this contract.
- Allowing `withdraw()` of the full amount at any time allows flexibility in case of problems.

WCN-17:Unused errors

Category	Severity	Code Reference	Status	Contributor
Code Style	**Informational**	• **code/contracts/lib/IWCNFTErrorCodes.sol#L9** • **code/contracts/lib/IWCNFTErrorCodes.sol#L12** • **code/contracts/lib/IWCNFTErrorCodes.sol#L15** • **code/contracts/lib/IWCNFTErrorCodes.sol#L18** • **code/contracts/lib/IWCNFTErrorCodes.sol#L24**	**Declined**	**Yaodao**

Code

```
9:    error ExceedsMaximumTokensPerTransaction();

12:    error ExceedsMaximumSupply();

15:    error ExceedsReserveSupply();

18:    error PublicSaleIsNotActive();

24:    error WrongETHValueSent();
```

Description

Yaodao : These errors are declared but never used in the contract.

Recommendation

Yaodao : Recommend removing these dead code.

Client Response

- We have left this as is since we utilize this file for other contracts

WCN-18:Unused internal function

Category	Severity	Code Reference	Status	Contributor
Code Style	**Informational**	• **code/contracts/Mythos.sol#L434-L441**	**Fixed**	**Yaodao, rajatbeladiya**

Code

```
434:    function _burn(uint256 tokenId, bool approvalCheck)
435:        internal
436:        virtual
437:        override
438:    {
439:        super._burn(tokenId, approvalCheck);
440:        _resetTokenRoyalty(tokenId);
441:    }
```

Description

Yaodao : The `_burn()` function is declared as the internal function which only can be called by the contract. However, the contract has no functions call the `_burn()` function.
rajatbeladiya : `_burn()` is used to burn token id and clears the royalty information for the token but the implementation of `_burn()` in `Mythos.sol` has internal visibility and is not used anywhere in the contract. so the `_burn()` function will not work if you intend to burn NFT token id.

Recommendation

Yaodao : Recommend removing the `_burn()` function.
rajatbeladiya : change visibility external from internal if you intend to burn NFT token id or remove it

Client Response

- refer to WCN-13

WCN-19:Using `constant` to save gas

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/contracts/lib/ERC721AOpensea.sol#L97	Acknowledged	Yaodao

Code

```
97:        return operator == address(0x1E0049783F008A0085193E00003D00cd54003c71);
```

Description

Yaodao : In solidity, using address public constant to store address saves gas. This is because constants are pre-hashed at compile time and don't need to be hashed at runtime. And if the hash operation is performed at runtime, it will consume more gas.

Consider below codes:

```
    function _isPriorityOperator(address operator)
        internal
        pure
        override
        returns (bool)
    {
        // OpenSea Seaport Conduit:
        // https://etherscan.io/address/0x1E0049783F008A0085193E00003D00cd54003c71
        // https://goerli.etherscan.io/address/0x1E0049783F008A0085193E00003D00cd54003c71
        return operator == address(0x1E0049783F008A0085193E00003D00cd54003c71);
    }
```

Recommendation

Yaodao : Recommend using a public constant to store the `address(0x1E0049783F008A0085193E00003D00cd54003c71)` and using it to check `operator`.

Consider below fix in the `ERC721AOpensea._isPriorityOperator()` function

```
    address public constant OPERATOR_ADDRESS = address(0x1E0049783F008A0085193E00003D00cd54003c71);
    function _isPriorityOperator(address operator)
        internal
        pure
        override
        returns (bool)
    {
        // OpenSea Seaport Conduit:
        // https://etherscan.io/address/0x1E0049783F008A0085193E00003D00cd54003c71
        // https://goerli.etherscan.io/address/0x1E0049783F008A0085193E00003D00cd54003c71
        return operator == OPERATOR_ADDRESS;
    }
```

Client Response

- refer to WCN-2

WCN-20: `WCNFTToken::setDefaultRoyalty` and `setTokenRoyalty` need more limit

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/contracts/lib/WCNFTToken.sol#L37 • code/contracts/lib/WCNFTToken.sol#L50-L56 • code/contracts/lib/WCNFTToken.sol#L55	Acknowledged	rajatbeladiya, zaskoh

Code

```
37:        _setDefaultRoyalty(receiver, feeNumerator);

50:    function setTokenRoyalty(
51:        uint256 tokenId,
52:        address receiver,
53:        uint96 feeNumerator
54:    ) external onlyRole(SUPPORT_ROLE) {
55:        _setTokenRoyalty(tokenId, receiver, feeNumerator);
56:    }

55:        _setTokenRoyalty(tokenId, receiver, feeNumerator);
```

Description

rajatbeladiya : `WCNFTToken.sol` implemented `ERC2981` to charge royalty amount on NFT transfer. `setTokenRoyalty()` is used to set the token royalty amount on transfer. Here it doesn't have any limit, it could be set to 100% which can lead to nft loss for the user.

zaskoh : The `feeNumerator` in `WCNFTToken` is used to set the fees that the protocol is charging / expecting when an NFT is sold (for example on Opensea).

Currently it's only checked if the value that is set by the admin is `feeNumerator <= _feeDenominator()` (<= 10000).

The fees can be set to a value like 9999 charging more than 99% for a sale.

Recommendation

rajatbeladiya : set limit to token royalty

zaskoh : Add a maximum fee that can be charged by the protocol and mention this in your documentation / site.

Example for a maximum of 30%:

```
+    uint96 MAX_FEE = 3000;
+    error FeeToHighForRoyalty();
@@ -34,6 +35,9 @@ contract WCNFTToken is WCNFTAccessControl, Ownable, ERC2981 {
         external
         onlyRole(SUPPORT_ROLE)
     {
+        if(feeNumerator > MAX_FEE) {
+            revert FeeToHighForRoyalty();
+        }
         _setDefaultRoyalty(receiver, feeNumerator);
     }

     @@ -52,6 +56,9 @@ contract WCNFTToken is WCNFTAccessControl, Ownable, ERC2981 {
         address receiver,
         uint96 feeNumerator
     ) external onlyRole(SUPPORT_ROLE) {
+        if(feeNumerator > MAX_FEE) {
+            revert FeeToHighForRoyalty();
+        }
         _setTokenRoyalty(tokenId, receiver, feeNumerator);
     }
```

Client Response

- The point is justified; we feel that setting a high fee would not be benefical to WCNFT or Mythos as there would be backlash from the community.

WCN-21:critical parameter should not change once auction is started

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/contracts/lib/SteppedDutchAuction.sol#L105-L119 • code/contracts/lib/SteppedDutchAuction.sol#L124-L128 • code/contracts/Mythos.sol#L184-L193	Acknowledged	rajatbeladiya

Code

```
    /**
     * @dev if a Dutch auction was paused using _endAuction it can be
     *  resumed with this function. No time is added to the duration so all
     *  elapsed time during the pause is lost.
     *
     * To restart a stopped Dutch auction from the startPrice with its full
     * duration, use _startAuction() again.
     */
    function _resumeAuction() internal virtual {
        if (startTime == 0) revert DutchAuctionHasNotStarted();
        if (auctionActive) revert DutchAuctionIsActive();

        auctionActive = true; // resume the auction
        emit DutchAuctionStart(startTime, duration);
    }

    function _endAuction() internal virtual isAuctionActive {
        auctionActive = false;

        emit DutchAuctionEnd(block.timestamp);
    }

    function createNewAuction(
        uint256 startPrice,
        uint256 finalPrice,
        uint256 priceStep,
        uint256 timeStepSeconds
    ) external onlyRole(SUPPORT_ROLE) {
        if (auctionActive) revert DutchAuctionIsActive();

        _createNewAuction(startPrice, finalPrice, priceStep, timeStepSeconds);
    }
```

Description

rajatbeladiya : `createNewAuction()` is used to create new auction with parameters like `startPrice`, `finalPrice`, `priceStep` and `timeStepSeconds`. It should be not changed during the auction. There can be scenario that it will let you change the parameters.

createNewAuction() has one check that if `auctionActive == false`, it will let you create the auction again.

`_resumeAuction()` is used to resume the auction and it has a description as below mentioned, which shows that pause will be achieved by `_endAuction()`

```
/**
     * @dev if a Dutch auction was paused using _endAuction it can be
     *  resumed with this function. No time is added to the duration so all
     *  elapsed time during the pause is lost.
     *
     * To restart a stopped Dutch auction from the startPrice with its full
     * duration, use _startAuction() again.
     */
```

The team can change the above variables during the sale. It will either increase or decrease the price of an NFT.

Scenario:

1. Auction started
2. during the auction, pausing the auction with `endAuction()` function will set `auctionActive` to false.
3. `createNewAuction()` can be used to set the auction parameters again
4. resume auction using `_resumeAuction()`

Recommendation

rajatbeladiya : prevent `createNewAuction()` during auction's pause state

Client Response

- We intentionally left this in case of user error. Refer to WCN-9

WCN-22:getRemainingSaleTime() returns wrong value if auction is not started

Category	Severity	Code Reference	Status	Contributor
Logical	**Informational**	• **code/contracts/lib/SteppedDutchAuction.sol#L147-L157**	**Declined**	**rajatbeladiya**

Code

```
147:    function getRemainingSaleTime() external view returns (uint256) {
148:        if (startTime == 0) {
149:            // not started yet
150:            return duration;
151:        } else if (_getElapsedAuctionTime() >= duration) {
152:            // already at the resting price
153:            return 0;
154:        }
155:
156:        return (startTime + duration) - block.timestamp;
157:    }
```

Description

rajatbeladiya : `getRemainingSaleTime()` is used to return the remaining auction sale duration. but when the auction is created and the auction is not started then it will return the `duration` of the auction. It is misleading, It should be returned 0 until the auction is started.

Recommendation

rajatbeladiya : return 0 until `startTime == 0`

Client Response

- We would like for the functionality to remain the same; the comments reflect what our intentions are. If it is 0, it might be confusing because it could be ended or not.

WCN-23:malicious user can prevent the owner to refund funds to users

Category	Severity	Code Reference	Status	Contributor
Logical	**Low**	• **code/contracts/Mythos.sol#L349-L359**	**Fixed**	**rajatbeladiya**

Code

```
349:    function sendTokensAndRefundBatch(ReceiverData[] calldata receivers)
350:        external
351:        onlyOwner
352:        nonReentrant
353:    {
354:        if (price == 0) revert PriceHasNotBeenSet();
355:
356:        for (uint256 i; i < receivers.length; i++) {
357:            _sendTokensAndRefund(receivers[i]);
358:        }
359:    }
```

Description

rajatbeladiya : the owner can send refund funds to users after the auction ended using `sendTokensAndRefundBatch()` to a batch of addresses. for that, it is looping over the receiver's addresses and sending eth to users' addresses. Suppose one of the receiver's addresses includes a malicious contract address which is implemented below code to revert the whole `sendTokensAndRefundBatch()` transaction.

```
contract Attack {

    fallback() payable external {
        while(true){

        }
    }

}
```

when the owner will send funds to users, this contract's exploit will use all the transaction gas and will revert the transaction, and the used gas will be lost (it could cost a lot to the owner).

there is a `sendTokensAndRefund()` function to send individually refund funds, but the impact will be the same the owner could lose a lot of gas.

`Impact:` the owner will be lost maximum gas with the transaction (it could cost a lot to the owner)

Recommendation

rajatbeladiya : implement refunds in a way that users can claim refunds by themselves. forward limited gas with the call() transaction by specifying gas.

```
(bool success, ) = to.call{value: refundValue, gas: 30000}("");
            if (!success) revert RefundFailed(to);
```

Client Response

- We have altered this function as recommended by another audit company (limit gas)
- We have also changed the RefundFailed to an event, so that the refund may continue in the event of a failed refund

WCN-24:pausing the auction will skip the auction's time duration

Category	Severity	Code Reference	Status	Contributor
Logical	**Low**	• **code/contracts/lib/SteppedDutchAuction.sol#L105-L119** • **code/contracts/lib/SteppedDutchAuction.sol#L124-L128**	**Acknowledged**	**rajatbeladiya**

Code

```
105:    /**
106:     * @dev if a Dutch auction was paused using _endAuction it can be
107:     *  resumed with this function. No time is added to the duration so all
108:     *  elapsed time during the pause is lost.
109:     *
110:     * To restart a stopped Dutch auction from the startPrice with its full
111:     * duration, use _startAuction() again.
112:     */
113:    function _resumeAuction() internal virtual {
114:        if (startTime == 0) revert DutchAuctionHasNotStarted();
115:        if (auctionActive) revert DutchAuctionIsActive();
116:
117:        auctionActive = true; // resume the auction
118:        emit DutchAuctionStart(startTime, duration);
119:    }

124:    function _endAuction() internal virtual isAuctionActive {
125:        auctionActive = false;
126:
127:        emit DutchAuctionEnd(block.timestamp);
128:    }
```

Description

rajatbeladiya : `_resumeAuction()` is used to resume the auction and it has a description as below mentioned, which shows that pause will be achieved by `_endAuction()`

```
/**
     * @dev if a Dutch auction was paused using _endAuction it can be
     *  resumed with this function. No time is added to the duration so all
     *  elapsed time during the pause is lost.
     *
     * To restart a stopped Dutch auction from the startPrice with its full
     * duration, use _startAuction() again.
     */
```

there is no any time maintaining regarding pausing the auction. So pausing an auction during the launch will skip the auction's actual time.

Scenario:

1. Auction created for 10 hours
2. 3 hours passed
3. Paused the auction
4. Resume the auction after 3 hours
5. Now users will have only 4 hours instead of 7 hours to purchase NFTs

Recommendation

rajatbeladiya : maintain the paused auction time

Client Response

- The intention of this function was only to resume the auction if a user has accidentally ended the function with `endAuction`.
- We have reworded the comment of the function as follows

```
    /**
     * @dev if a Dutch auction was ended prematurely using _endAuction it can be
     *  resumed with this function. No time is added to the duration so all
     *  elapsed time during the pause is lost.
     *
```

WCN-25:preBuyActive can't be set to same value

Category	Severity	Code Reference	Status	Contributor
Logical	**Informational**	• **code/contracts/Mythos.sol#L131**	**Declined**	**parth_15**

Code

```
131:    function setPreBuyActive(bool state) external onlyRole(SUPPORT_ROLE) {
```

Description

parth_15 : `setPreBuyActive` function is used to set the value of `preBuyActive` and can only be called by `SUPPORT_ROLE`. It first checks that whether auction is not active and price is zero before setting the value of `preBuyActive`. It is possible to call `setPreBuyActive` with same value and setting the same value. There is no sanity check to prevent `SUPPORT_ROLE` to setting it to same value.

Recommendation

parth_15 : Add a sanity check that if the `SUPPORT_ROLE` is trying to set `preBuyActive` to same value, the call gets reverted.

Consider below fix in the `setPreBuyActive` function

```
function setPreBuyActive(bool state) external onlyRole(SUPPORT_ROLE) {
        if(preBuyActive == state) revert PreBuyActiveIsSame();
        if (auctionActive) revert DutchAuctionIsActive();
        if (price != 0) revert PriceHasBeenSet();
        preBuyActive = state;
    }
```

Client Response

- We don't believe adding this will provide any value; should not affect contract operation
- Only costs the SUPPORT_ROLE user gas for executing the function

WCN-26:provenance is not used in the contract

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	**Informational**	• **code/contracts/Mythos.sol#L36** • **code/contracts/Mythos.sol#L404**	**Acknowledged**	**parth_15**

Code

```
36:    string public provenance;

404:    function setProvenance(string memory provenance_)
```

Description

parth_15 : There is a state variable named `provenance` and function `setProvenance` to set it. This values are not used anywhere in the contract and thus can be removed from the contract.

The impact is that adding function for setting a value of unused variable can increase contract size.

Recommendation

parth_15 : Remove unused variables from the contract.

Client Response

- We use this function to provide customers with a way to verify the randomness of the nfts.

WCN-27:setMinimumPreBuyPrice lacks sanity check

Category	Severity	Code Reference	Status	Contributor
Logical	**Low**	• **code/contracts/Mythos.sol#L142**	**Acknowledged**	**parth_15**

Code

```
142:    function setMinimumPreBuyPrice(uint256 minimumPreBuyPriceInWei)
```

Description

parth_15 : `setMinimumPreBuyPrice` sets the `minimumPreBuyPrice`. It lacks sanity check and can be set to `0` which implies free mints.

The impact is user can make any small bids during the preBuy phase.

Recommendation

parth_15 : Use a sanity check to ensure that `minimumPreBuyPrice` is greater than `0` before setting the value.

Consider below fix in the `setMinimumPreBuyPrice` function

```
function setMinimumPreBuyPrice(
        uint256 minimumPreBuyPriceInWei
    ) external onlyRole(SUPPORT_ROLE) {
        if(minimumPreBuyPriceInWei == 0) revert InvalidValue();
        minimumPreBuyPrice = minimumPreBuyPriceInWei;
    }
```

Client Response

- The `minimumPreBuyPrice` is decided by the client and set by `SUPPORT_ROLE`.
- It is unlikely that the preBuy phase would be started with the wrong `minimumPreBuyPrice` set.
    - It will be set and checked before opening for deposits.

WCN-28:users may lose bid amount

Category	Severity	Code Reference	Status	Contributor
Logical	Informational	• code/contracts/Mythos.sol#L241	Acknowledged	rajatbeladiya

Code

```
241:        bidder.contribution = uint216(contribution_);
```

Description

rajatbeladiya : users can place a bid in ETH or add to their existing bid using the `bid()` function. In the `bid()` function it is typecasting the user's `contribution` from uint256 to uint216. Here users can bid any amount, it could be greater than uint216. It will lead to a loss amount for the user if the amount is greater than the uint216 max value. The likelihood is low but worth mentioning it.

Recommendation

rajatbeladiya : remove typecasting `uint216(contribution_)`

Client Response

- The maximum size of `uint216` is `2^216 - 1 = 1x10^65`,
- This means a user's contribution would be truncated if it exceeded approx `1x10^65 WEI` or `1x10^47 ETH`.
- The entire circulating supply of ETH is less than this, at approx `1.2x10^8 ETH`

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you (“Customer” or the “Company”) in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Secure3’s prior written consent in each instance.

This report is not an “endorsement” or “disapproval” of any particular project or team. This report is not an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3’s position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.