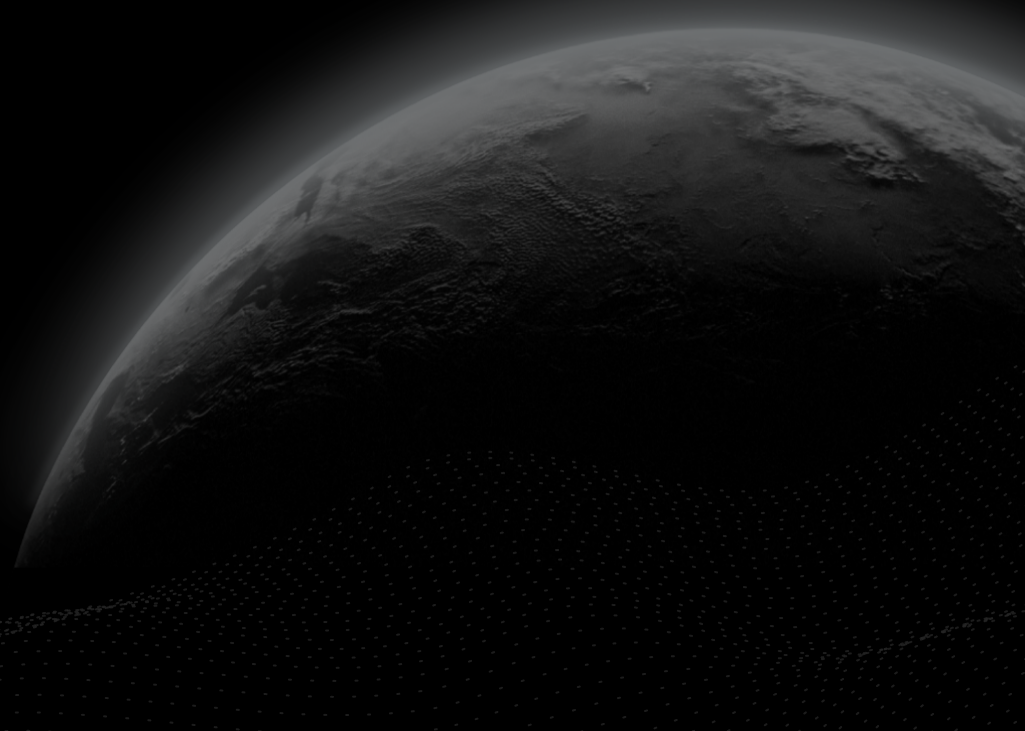
CERTIK

Security Assessment

Co-Museum

CertiK Verified on Nov 16th, 2022



CertiK Verified on Nov 16th, 2022

Co-Museum

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
ERC-20	Ethereum	Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 11/16/2022	N/A

CODEBASE

https://github.com/co-museum

...View All

COMMITS

- 50e46535a3da8561f4cbe17b6a42ae9d69bfd77c
- ba58ea2e4c52c08a86e6aff28200f32a69ca366f
- a6c08ed41af5ad4fae6223961148bbd4980ffa33

...View All

Vulnerability Summary

26	23	0	0	3	0	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined	Unresolved

2	Critical	2 Resolved	Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
4	Major	2 Resolved, 2 Acknowledged	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
2	Medium	2 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
4	Minor	4 Resolved	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
14	Informational	13 Resolved, 1 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | CO-MUSEUM

CODEBASE | CO-MUSEUM

Repository

https://github.com/co-museum

Commit

- 50e46535a3da8561f4cbe17b6a42ae9d69bfd77c
- ba58ea2e4c52c08a86e6aff28200f32a69ca366f
- a6c08ed41af5ad4fae6223961148bbd4980ffa33
- ab3844b62cbc6864c38eb3be4e794da225f15433

AUDIT SCOPE | CO-MUSEUM

0 files audited

ID	Repo	File	SHA256 Checksum

APPROACH & METHODS | CO-MUSEUM

This report has been prepared for Co-Museum to discover issues and vulnerabilities in the source code of the Co-Museum project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from major to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

REVIEW NOTES | CO-MUSEUM

Overview

Co-Museum is a new cultural institution devoted to democratising ownership of some of the world's masterpieces, protecting art in purpose-built viewing facilities, and creating a fully liquid and permissionless art market.

Privileged Functions

In `AllowanceCrowdsale.sol`, the owner has authority over the following functions:

- `AllowanceCrowdsale.startSale()`: Start the sale for a batch of $ART tokens and/or associated NFTs;
- `AllowanceCrowdsale.stopSale()`: Stop the sale for a batch of $ART tokens and/or associated NFTs;
- `Ownable.transferOwnership()`: Change the owner of the contract;
- `Ownable.renounceOwnership()`: Renounce ownership of the contract.

In `ERC721VaultFactory.sol`, the owner of the contract has the authority over the following functions:

- `ERC721VaultFactory.pause()`: Pause the contract;
- `ERC721VaultFactory.unpause()`: Unpause the contract;
- `Ownable.transferOwnership()`: Change the owner of the contract;
- `Ownable.renounceOwnership()`: Renounce ownership of the contract.

Any compromise to the owner account in `AllowanceCrowdsale.sol` and `ERC721VaultFactory.sol` will enable the malicious actor to take advantage of this authority and disable the contract permanently.

In `Settings.sol`, the owner of the contract has the authority over the following functions:

- `Settings.setMaxAuctionLength`: Set the maximum length of auction;
- `Settings.setMinAuctionLength`: Set the minimum length of auction;
- `Settings.setGovernanceFee`: Set the rate of governance fee;
- `Settings.setMaxCuratorFee`: Set the maximum rate of curator fee;
- `Settings.setMinBidIncrease`: Set the minimum increase rate of bid;
- `Settings.setMinVotePercentage`: Set the minimum percentage of token required to be voting for an auction to start;
- `Settings.setMaxReserveFactor`: Set the maximum percentage increase over the initial reserve amount;
- `Settings.setMinReserveFactor`: Set the maximum percentage decrease over the initial reserve amount;
- `Settings.setFeeReceiver`: Set the address to receive the governance fee;
- `Ownable.renounceOwnership`: Remove the owner from the contract;
- `Ownable.transferOwnership`: Change the owner of the contract.

Any compromise to the owner account in `Settings.sol` will enable the malicious actor to take advantage of the authority to earn fees and/or force an auction to end.

In `ERC721MembershipUpgradeable.sol`, the owner of the contract has the authority over the following functions:

- `ERC721MembershipUpgradeable.setBaseURI`: Change the variable `_membershipBaseURI`;
- `ERC721MembershipUpgradeable.setDefaultRoyalty`: Set the royalty information that all ids default to;
- `ERC721MembershipUpgradeable.deleteDefaultRoyalty`: Remove default royalty information.

Any compromise to the owner account in `ERC721MembershipUpgradeable` will enable the malicious user to alter royalty information and change the base URI.

In `VoteDelegator.sol`, the owner of the contract has the authority over the following functions:

- `VoteDelegator.updateUserPrice`: Update the desired sale price;
- `VoteDelegator.withdraw`: Transfer the token owned by this contract to a designated address;
- `Ownable.transferOwnership`: Change the owner of the contract;
- `Ownable.renounceOwnership`: Renounce ownership of the contract.

Any compromise to the owner account in `VoteDelegator.sol` will enable the malicious user to steal the tokens owned by this contract.

To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should be also considered to move to the execution queue of `Timelock` contract.

FINDINGS | CO-MUSEUM

26	2	4	2	4	14
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Co-Museum. Through this audit, we have uncovered 26 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
GLOBAL-01	**Centralization Related Risks**	**Centralization / Privilege**	**Major**	**Acknowledged**
GLOBAL-02	**Centralized Control Of Contract Upgrade**	**Centralization / Privilege**	**Major**	**Acknowledged**
ACB-01	Lack Of Access Control Over `setRates()` Function	Volatile Code	Critical	Resolved
ACB-02	Potential Reentrancy Attack In BuyNFTs()	Volatile Code	Major	Resolved
ACB-03	Remaining `eth` Is Not Sent Back To Users	Logical Issue	Medium	Resolved
ACB-04	Usage Of `transfer()` For Sending Ether	Volatile Code	Minor	Resolved
ERC-01	Lack Of Access Control Over `redeem` Function	Volatile Code	Critical	Resolved
ERC-02	Multiplies Vulnerabilties In _redeem()	Control Flow, Language Specific	Major	Resolved
ERC-03	Incorrect Variable Usage	Inconsistency	Minor	Resolved
ERT-01	Potential Reentrancy Attack	Volatile Code	Medium	Resolved

ID	Title	Category	Severity	Status
ERT-02	Anyone Can Call `redeem` When `_supply` Is Initialized As 0	Logical Issue	Minor	Resolved
SET-01	Incorrect Variable Usage	Inconsistency	Minor	Resolved
ACB-05	`uint8` Type Can Be Changed To `uint256`	Volatile Code	Informational	Resolved
ACB-06	Incorrect Grammar	Coding Style	Informational	Resolved
CON-01	Typo	Typo	Informational	Resolved
ERC-04	Missing Input Validation	Volatile Code	Informational	Resolved
ERT-03	Unused Variable	Coding Style	Informational	Resolved
ERT-04	Missing Emit Events	Coding Style	Informational	Resolved
ERT-05	Incorrect Comments	Inconsistency	Informational	Resolved
ERT-06	Lack Of Input Validation	Logical Issue	Informational	Resolved
ERT-07	Uninformative Event	Coding Style	Informational	Resolved
ERT-08	Misleading Comments	Inconsistency	Informational	Resolved
ERT-09	Unable To Redeem After Claiming Fees	Logical Issue	Informational	Acknowledged
ERV-01	Lack Of Natspec Comments	Coding Style	Informational	Resolved
NFT-01	Missing Emit Events	Coding Style	Informational	Resolved

ID	Title	Category	Severity	Status
SET-02	Lack Of Input Validation	Coding Style	Informational	Resolved

GLOBAL-01 | CENTRALIZATION RELATED RISKS

Category	Severity	Location	Status
Centralization / Privilege	**Major**		**Acknowledged**

Description

In `AllowanceCrowdsale.sol` the owner has authority over the following functions:

- `AllowanceCrowdsale.startSale()` : Start the sale for a batch of $ART tokens and/or associated NFTs
- `AllowanceCrowdsale.stopSale()` : Stops the sale for a batch of $ART tokens and/or associated NFTs
- `Ownable.transferOwnership()` : Change the owner of the contract
- `Ownable.renounceOwnership()` : Remove the owner of the contract

In `ERC721VaultFactory.sol` , the owner of the contract has the authority over the following functions:

- `ERC721VaultFactory.pause()` : Pause the contract
- `ERC721VaultFactory.unpause()` : Unpause the contract
- `Ownable.transferOwnership()` : Change the owner of the contract
- `Ownable.renounceOwnership()` : Remove the owner from the contract

Any compromise to the owner account in `AllowanceCrowdsale.sol` and `ERC721VaultFactory.sol` will enable the malicious actor to take advantage of this authority and disable the contract permanently.

In `Settings.sol` , the owner of the contract has the authority over the following functions:

- `Settings.setMaxAuctionLength` : Set the maximum length of auction
- `Settings.setMinAuctionLength` : Set the minimum length of auction
- `Settings.setGovernanceFee` : Set the rate of governance fee
- `Settings.setMaxCuratorFee` : Set the maximum rate of curator fee
- `Settings.setMinBidIncrease` : Set the minimum increase rate of bid
- `Settings.setMinVotePercentage` : Set the minimum percentage of token required to be voting for an auction to start
- `Settings.setMaxReserveFactor` : Set the maximum percentage increase over the initial reserve amount
- `Settings.setMinReserveFactor` : Set the maximum percentage decrease over the initial reserve amount
- `Settings.setFeeReceiver` : Set the address to receive the governance fee
- `Ownable.renounceOwnership` : Remove the owner from the contract
- `Ownable.transferOwnership` : Change the owner of the contract

Any compromise to the owner account in `Settings.sol` will enable the malicious actor to take advantage of the authority to earn fees and/or force an auction to end.

In `ERC721MembershipUpgradeable.sol`, the owner of the contract has the authority over the following functions:

- `ERC721MembershipUpgradeable.setBaseURI`: Change the variable `_membershipBaseURI`
- `ERC721MembershipUpgradeable.setDefaultRoyalty`: Sets the royalty information that all ids default to
- `ERC721MembershipUpgradeable.deleteDefaultRoyalty`: Remove default royalty information

Any compromise to the owner account in `ERC721MembershipUpgradeable` will enable the malicious user to alter royalty information and change the base URI.

In `VoteDelegator.sol`, the owner of the contract has the authority over the following functions:

- `VoteDelegator.updateUserPrice`: Update the desired sale price
- `VoteDelegator.withdraw`: Transfer the token owned by this contract to a designated address
- `Ownable.transferOwnership`: Change the owner of the contract
- `Ownable.renounceOwnership`: Remove the owner from the contract

Any compromise to the owner account in `VoteDelegator.sol` will enable the malicious user to steal the tokens owned by this contract.

In commit ab3844b62cbc6864c38eb3be4e794da225f15433, `ERC721ArtNFT` and `ERC721HonoraryMembership` extends `DefaultOperatorFilterer`, `ERC721MembershipUpgradeable` extends `DefaultOperatorFiltererUpgradeable`. These changes add restrictions on transfer operations and only the user who has `AllowedOperator` role can call the following functions:

- `transferFrom(address from,address to,uint256 tokenId)`
- `safeTransferFrom(address from,address to,uint256 tokenId)`
- `safeTransferFrom(address from,address to,uint256 tokenId,bytes memory data)`

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[Co-Museum]: The team will be using a nested gnosis multisig for both treasury and owner wallets (2/2 comprised out of a 2/2 and a 2/3 wallet) and will publish the relevant addresses on Medium and use `@openzeppelin/contracts/governance/TimelockController.sol`.

GLOBAL-02 | CENTRALIZED CONTROL OF CONTRACT UPGRADE

Category	Severity	Location	Status
Centralization / Privilege	**Major**		**Acknowledged**

Description

The following should be upgradeable contracts, the owner can upgrade the contract without the community's commitment. If an attacker compromises the account, he can change the implementation of the contract and drain tokens from the contract.

- `TokenVault()`;
- `PartiallyPausableUpgradeable()`;
- `ERC721MembershipUpgradeable()`;
- `VoteDelegator()`.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[Co-Museum]: The team will be using a nested gnosis multisig for both treausury and owner wallets (2/2 compirsed out of a 2/2 and a 2/3 wallet) and will publish the relevant addresses on Medium and use `@openzeppelin/contracts/governance/TimelockController.sol`.

ACB-01 LACK OF ACCESS CONTROL OVER `setRates()` FUNCTION

Category	Severity	Location	Status
Volatile Code	Critical	crowdsale/AllowanceCrowdsale.sol: 101	Resolved

Description

The function `setRates()` does not contain any access control and allows any user to change the value of `stablecoinRate` and `ethRate`. Furthermore, there is no minimum or maximum threshold for both values.

Exploit Scenario

- An attacker can call `setRates()` function to set the smallest `ethRate`, for example, 1 wei. And then he can buy NFT at an extremely low price.

Recommendation

Since the corresponding variables are used for sensitive operations such as transferring tokens, consider implementing access control and set a minimum and/or maximum threshold for both values.

Alleviation

[Co-Museum]: The team resolved this issue in commit a6c08ed41af5ad4fae6223961148bbd4980ffa33 by adding `onlyOwner` modifier to the function.

ACB-02 | POTENTIAL REENTRANCY ATTACK IN BUYNFTS()

Category	Severity	Location	Status
Volatile Code	Major	crowdsale/AllowanceCrowdsale.sol: 163	Resolved

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

The function `buyNFTs()` is vulnerable to the reentrancy attack :

```
178
ERC721MembershipUpgradeable(membershipContract).redeem(whitelist.tierCode,tokenHoldi
ngWallet, msg.sender);
```

The attacker will be capable to buy more NFTs than his current allocation, using the following process:

Exploit Scenario

1. The attacker calls `buyNFTs()`.
2. The contract calls `redeem(whitelist.tierCode, tokenHoldingWallet, msg.sender);` who calls `_safeMint()`
3. `SafeMint()` will jump back into the attacker contract **without** updating the `claimed[msg.sender] = true;`
4. The attacker calls again `buyNFTs()`, and go back to Step 1.

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

[Co-Museum]: The team resolved this issue in commit 6c494deaeb0c2d48e53bad3745fce46598a3c74f by using using the [Checks-Effects-Interactions Pattern].

ACB-03 | REMAINING `eth` IS NOT SENT BACK TO USERS

Category	Severity	Location	Status
Logical Issue	Medium	crowdsale/AllowanceCrowdsale.sol: 244	Resolved

Description

When the `eth` sent by the user in `_receivePayment()` function is greater than `quantity * ethRate`, the remaining part will not be returned to the user.

Recommendation

Consider returning back the remaining `eth` in `_receivePayment()` function. An example of code is provided below.

```
require(msg.value >= quantity * ethRate, "crowdsale:not enough eth");
uint256 back = msg.value - quantity * ethRate;
if(back>0){
    (bool success, ) = msg.sender.call{ value: back }("");
    require(success, "unable to send value");
}
treasuryWallet.transfer(quantity * ethRate);
```

Alleviation

[Co-Museum]: The team resolved this issue in commit 30feb3fe39941f817523ddef005455742e29a3bd by sending back the remaining ETH to the user.

ACB-04 | USAGE OF `transfer()` FOR SENDING ETHER

Category	Severity	Location	Status
Volatile Code	Minor	crowdsale/AllowanceCrowdsale.sol: 226	Resolved

Description

After EIP-1884 was included in the Istanbul hard fork, it is not recommended to use `.transfer()` or `.send()` for transferring ether as these functions have a hard-coded value for gas costs making them obsolete as they are forwarding a fixed amount of gas, specifically `2300`. This can cause issues in case the linked statements are meant to be able to transfer funds to other contracts instead of EOAs.

Recommendation

We advise that the linked `.transfer()` and `.send()` calls are substituted with the utilization of the `sendValue()` function from the `Address.sol` implementation of OpenZeppelin either by directly importing the library or copying the linked code.

Alleviation

[Co-Museum]: The team resolved this issue in commit 30feb3fe39941f817523ddef005455742e29a3bd by using the the sendValue() function from the Address.sol.

ERC-01 | LACK OF ACCESS CONTROL OVER `redeem` FUNCTION

Category	Severity	Location	Status
Volatile Code	Critical	nfts/ERC721MembershipUpgradeable.sol: 263	Resolved

Description

The function `redeem()` mints an NFT to the address specified in `nftTo`. However since it does not have any access control, any user can call this function and designate themselves as `nftTo` and receive the NFT.

Exploit Scenario

- The attacker calls `redeem()` with `nftTo` set has the attacker's address and `tierCode` has a valid `TierCode`.
- In the end, the NFT will be minted and transferred to the attacker's address.

Recommendation

If this is not the intended functionality, it is recommended to add an access control limiting the users who can call redeem or users who meet certain criteria.

Alleviation

Updated on 2022-09-30 from Co-Museum:
Anyone can call redeem, but they need to have ERC20 tokens minted by the TokenVault at a particular address. The attacker will not have these tokens unless it has been transferred or sold to them by the Co-Museum.

In particular, we have the following lines:

```
require(TokenVault(vault).balanceOf(erc20From) >= tier.price,
"membership:insufficient balance");
TokenVault(vault).transferFrom(erc20From, address(this), tier.price);
```

Clearly, the caller must have the token minted by the tokenVault, in which case Co-Museum allows them to redeem a NFT.

Reply from Certik:
Malicious hackers may observe the pending transaction which will execute the `redeem` function, and launch a similar transaction but with the hacker's address of `nftTo`, and the NFT will be minted and transferred to the attacker's address. We recommended adding a check `require(erc20From==msg.sender)` to the `redeem` function.

[Co-Museum]: The team resolved this issue in commit 8109670bc859b0ef28ac8d5555950209ab4067b1 by adding an access control.

ERC-02 | MULTIPLIES VULNERABILTIES IN _REDEEM()

Category	Severity	Location	Status
Control Flow, Language Specific	Major	nfts/ERC721MembershipUpgradeable.sol: 271	Resolved

Description

The function `_redeem()` will redeem membership associated with a certain `tierCode`.

- Because `_redeem()` is **external** without any control checks, anyone can call this function and could mint NFTs.
- `erc20From` is used without any control checks, anyone could create a malicious token and use it in the `_redeem()` function.

```
271  function _redeem(
272          TierCode tierCode,
273          address erc20From,
274          address nftTo
275      ) external {
276          uint256 id;
277          Tier storage tier = _getTierByCode(tierCode);
278
279          if (tier.releasedIds.length > 0) {
280              id = tier.releasedIds[tier.releasedIds.length - 1];
281              tier.releasedIds.pop();
282          } else {
283              require(tier.currId < tier.end, "membership:cannot mint more tokens
at tier");
284              id = tier.currId;
285              tier.currId++;
286          }
287          emit Redeem(nftTo, id);
288          _safeMint(nftTo, id);
289
290          require(TokenVault(vault).balanceOf(erc20From) >= tier.price,
"membership:insufficient balance");
291          TokenVault(vault).transferFrom(erc20From, address(this), tier.price);
292      }
```

Recommendation

The auditors recommend adding a control check using a `require()` to restrict access only to the crowdsale.

For example:

```
require(msg.sender == crowdsale_contract);
```

Also, the auditors recommend replacing the `external` visibility with `internal`.

For the `erc20From` controls check missing, we recommend using a whitelist for specific tokens.

Alleviation

Update on 2022-09-30 from Co-Museum:\

1. "erc20From is used without any control checks, anyone could create a malicious token and use it in the _redeem() function" - erc20From says which address it is that is transferring the associated $ART token to mint a membership NFT and the NFT is minted to nftTo address. It is not the address of the erc20 token required for redemption.
2. "anyone can call this function and could mint NFTs" - This is expected behaviour if someone has the token minted from the relevant tokenVault
3. "_reedeem is external" - We want to msg.sender to be the contract itself when calling redeem, therefore it is external. Is this a problem?

Reply from Certik:
The issue is the same as that mentioned in the finding ERC-01.

[Co-Museum]: The team resolved this issue in commit 8109670bc859b0ef28ac8d5555950209ab4067b1 by adding an access control.

ERC-03 | INCORRECT VARIABLE USAGE

Category	Severity	Location	Status
Inconsistency	Minor	nfts/ERC721MembershipUpgradeable.sol: 195, 210	Resolved

Description

The `genesisTier` and `friendTier` uses the input of the wrong tier for `releasedIds`. Though both `genesisTier` and `friendTier` are both empty during initialization so there are no impact to the logic of the code.

Recommendation

Change the code to following for improved clarity and legibility:

```
195  releasedIds: genesisIdStack
```

```
210  releasedIds: friendIdStack
```

Alleviation

[Co-Museum]: The team resolved this issue in commit 1c8ec3c22d97de172edd2fccb955cff9cf9a449f by using the correct variables.

ERT-01 | POTENTIAL REENTRANCY ATTACK

Category	Severity	Location	Status
Volatile Code	Medium	fractional/ERC721TokenVault.sol: 424	Resolved

Description

A reentrancy attack can occur depending on the implementation of the `token` contract. Since `IERC721(token).transferFrom()` is an external call, it can reenter the `end()` function to claim fees multiple times.

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

[Co-Museum]: The team resolved this issue in commit 2c919f5ecdc6712c81313c6f8970d6e8e37c3d08 by using the [Checks-Effects-Interactions Pattern].

ERT-02 ANYONE CAN CALL `redeem` WHEN `_supply` IS INITIALIZED AS 0

Category	Severity	Location	Status
Logical Issue	Minor	fractional/ERC721TokenVault.sol: 148	Resolved

Description

When initializing a `TokenVault`, some tokens will be minted to `_curator`:

```
function initialize(
        address _curator,
        address _token,
        uint256 _id,
        uint256 _supply,
        uint256 _listPrice,
        uint256 _fee,
        string memory _name,
        string memory _symbol,
        address _usdc
    ) external initializer {
        // initialize inherited contracts
        __ERC20_init(_name, _symbol);
        __ERC721Holder_init();
        __PartiallyPausableUpgradeable_init(Ownable(settings).owner());

        // set storage variables
        token = _token;
        id = _id;
        auctionLength = 3 days;
        curator = _curator;
        fee = _fee;
        lastClaimed = block.timestamp;
        auctionState = State.disabled;
        userPrices[_curator] = _listPrice;
        usdc = _usdc;

        _mint(_curator, _supply);
    }
```

There is no check if `_supply` is greater than 0. When `_supply` is mistakenly set to 0, the `totalSupply` will be 0. In this case, anyone can call `redeem()` to redeem the NFT because anyone can burn 0 tokens:

```
function redeem() external {
        require(auctionState == State.inactive, "redeem:no redeeming");
        _burn(msg.sender, totalSupply());

        // transfer erc721 to redeemer
        IERC721(token).transferFrom(address(this), msg.sender, id);

        auctionState = State.redeemed;

        emit Redeem(msg.sender);
}
```

Recommendation

Consider adding a check on the `_supply`:

```
require(_supply>0,"invalid supply");
```

Alleviation

[Co-Museum]: The team resolved this issue in commit 68facb3d7cbe9da7d3dce7b836b042fc611c2651 by adding the check.

SET-01 | INCORRECT VARIABLE USAGE

Category	Severity	Location	Status
Inconsistency	Minor	fractional/Settings.sol: 106	Resolved

Description

Despite the function `setMaxCuratorFee` updating the maximum threshold for curator fee, the event emits `governanceFee` instead of the `maxCuratorFee` .

Recommendation

Change the corresponding line to the following:

```
106  emit UpdateCuratorFee(maxCuratorFee, _fee);
```

Alleviation

[Co-Museum]: The team resolved this issue in commit 9d32ba67b1a4033da8be18c07bc701cef6131840 by using the correct variables.

ACB-05 | `uint8` TYPE CAN BE CHANGED TO `uint256`

Category	Severity	Location	Status
Volatile Code	Informational	crowdsale/AllowanceCrowdsale.sol: 145, 165	Resolved

Description

The for loop sets the variable `i` as a `uint8` data type. If whitelists.length is greater than 255, the function will always revert since `i` will overflow after 255 and code written with Solidity version greater than 0.8 prevents overflow from occurring.

Furthermore, the functions `buyNFTs()` takes an input parameter that has a data type of `uint8`. Again, if the length of the whitelist is longer than 255, items with index greater than 255 will not be accessible.

Recommendation

Unless it is certain that the length of whitelists will not exceed 255, consider changing the data type from `uint8` to `uint256`.

Alleviation

[Co-Museum]: The team resolved this issue in commit ee20b24087121197c20ecee06775b8c87a3d088b by changing the data type from `uint8` to `uint256`.

ACB-06 | INCORRECT GRAMMAR

Category	Severity	Location	Status
Coding Style	Informational	crowdsale/AllowanceCrowdsale.sol: 155	Resolved

Description

The comment "WhiltelistIndex represents which which whitelist" is grammatically incorrect and also contains a typo. It should be "WhitelistIndex represents which whitelist".

Recommendation

Consider changing the sentence as mentioned in the description for improved clarity and legibility.

Alleviation

[Co-Museum]: The team resolved this issue in commit b5e27858444f1d2b8fe54d4729e81c899e54b224 by fixing the grammar.

CON-01 | TYPO

Category	Severity	Location	Status
Typo	Informational	crowdsale/AllowanceCrowdsale.sol: 33, 36, 111, 150, 191; fractional/ERC721VaultFactory.sol: 35, 36	Resolved

Description

In `AllowanceCrowdsale.sol`:

- "their" is misspelled as "thier"
- "particular" is misspelled as "partcular"
- "membership" is misspelled as "membershio"

In `ERC721VaultFactory.sol`:

- "symbol" is misspelled as "sumbol"
- "of" is misspelled as "fo"

Recommendation

Consider correcting the typo for improved code legibility and clarity.

Alleviation

[Co-Museum]: The team resolved this issue in commit 0aec252d8fe6389659a7cc3f3134ab5e2544b690 by fixing typos.

ERC-04 | MISSING INPUT VALIDATION

Category	Severity	Location	Status
Volatile Code	Informational	nfts/ERC721MembershipUpgradeable.sol: 176	Resolved

Description

The input parameter `genesisEnd`, `foundationEnd`, `friendEnd` should be greater than each other in the respective order. However there is no check implemented to make sure it is the case.

Recommendation

Consider adding the following condition:

```
101  require(genesisEnd < foundationEnd && foundationEnd < friendEnd);
```

Alleviation

[Co-Museum]: The team resolved this issue in commit 7ae917273c307a62826b5cccc36da22bf0622941 by adding the check.

ERT-03 | UNUSED VARIABLE

Category	Severity	Location	Status
Coding Style	Informational	fractional/ERC721TokenVault.sol: 80	Resolved

Description

The variable `vaultClosed` is not used in the contract `ERC721TokenVault` nor any of the contract that imports `ERC721TokenVault.sol`.

Recommendation

Consider removing unnecessary variable for improved clarity and legibility.

Alleviation

[Co-Museum]: The team resolved this issue in commit 3fd70aad118bad9994c90bbd124524f0ab7f9014 by removing unnecessary variable.

ERT-04 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Coding Style	Informational	fractional/ERC721TokenVault.sol: 121, 151, 180, 209	Resolved

Description

The following function that affects the status of sensitive variables should emit events.

- `initialize()`
- `toggleAuctions()`
- `kickCurator()`
- `updateCurator()`

Recommendation

Consider changing the visibility of the corresponding functions to `external` for optimized gas cost.

Alleviation

[Co-Museum]: The team resolved this issue in commit 5b2ea19bf90aff21217e147de4463e9ebccfde79 by adding events.

ERT-05 | INCORRECT COMMENTS

Category	Severity	Location	Status
Inconsistency	Informational	fractional/ERC721TokenVault.sol: 216	Resolved

Description

The comments describe the `_length` parameter as "the new base price". However, it should be "the new auction length".

Recommendation

Consider changing the comments for improved legibility and clarity.

Alleviation

[Co-Museum]: The team resolved this issue in commit 023de14ab57cdbd5479505ce9c07a0e39d9e1824 by changing comments.

ERT-06 | LACK OF INPUT VALIDATION

Category	Severity	Location	Status
Logical Issue	Informational	fractional/ERC721TokenVault.sol: 142	Resolved

Description

There are no conditions checking whether `_fee` is greater than the `ISettings(settings).maxCuratorFee()` during initialization. This allows a curator to set an arbitrary amount for fees that can be greater than 100%.

Recommendation

Check if this is the intended design of the contract and if not to implement a check on the value of `_fee`.

Alleviation

[Co-Museum]: The team resolved this issue in commit 5df1c975a218c1fab8db50d05f7af43f7b1ead5d by adding the check.

ERT-07 | UNINFORMATIVE EVENT

Category	Severity	Location	Status
Coding Style	Informational	fractional/ERC721TokenVault.sol: 112	Resolved

Description

Despite fees being collected by both the curator and governance, the event only emits the amount collected.

Recommendation

Consider emitting the address that collected the fee in the event to distinguish how much fees were collected by curator and governance.

Alleviation

[Co-Museum]: The team resolved this issue in commit 6c740efbc8c359d7c382c7878a2446a24119ffc0 by emitting the address that collected the fee in the event.

ERT-08 | MISLEADING COMMENTS

Category	Severity	Location	Status
Inconsistency	Informational	fractional/ERC721TokenVault.sol: 394	Resolved

Description

The comments mention "msg.value is the bid amount". However this function is not `payable` and based on the implementation, the token to be sent is usdc instead of ether.

Recommendation

Consider removing the comment "The msg.value is the bid amount."

Alleviation

[Co-Museum]: The team resolved this issue in commit 023de14ab57cdbd5479505ce9c07a0e39d9e1824 by removing the comment.

ERT-09 | UNABLE TO REDEEM AFTER CLAIMING FEES

Category	Severity	Location	Status
Logical Issue	Informational	fractional/ERC721TokenVault.sol: 432~442	Acknowledged

Description

The `redeem()` function is designed to redeem NFT by burning all tokens, here is the implementation:

```
432  function redeem() external {
433        require(auctionState == State.inactive, "redeem:no redeeming");
434        _burn(msg.sender, totalSupply());
435
436        // transfer erc721 to redeemer
437        IERC721(token).transferFrom(address(this), msg.sender, id);
438
439        auctionState = State.redeemed;
440
441        emit Redeem(msg.sender);
442    }
```

This function can only be called successfully if the caller has all the tokens.

The `claimFees()` function is designed to claim fees for the curator and governance. After calling this function, some of the tokens will be minted to the governance, which means that no one can own all the tokens.

```
268  ...
269  if (curator != address(0)) {
270      _mint(curator, curatorMint);
271      emit FeeClaimed(curatorMint);
272  }
273  if (govAddress != address(0)) {
274      _mint(govAddress, govMint);
275      emit FeeClaimed(govMint);
276  }
277  ...
```

If someone calls the `claimFees()` function, then no one can call the `redeem()` function successfully because no one will own all the tokens at this time.

Recommendation

Consider redesigning the logic of `claimFees()` function. For example, disallow calling this function when the bid is inactive.

Alleviation

[Co-Museum]: The redeem function's primary purpose is for people to be able to change their mind immediately after minting. Once the ball gets rolling an auction is the only realistic way out. Unless they acquire the entire token supply the organic way (from governance/the curator as well) that is.

ERV-01 | LACK OF NATSPEC COMMENTS

Category	Severity	Location	Status
Coding Style	Informational	fractional/ERC721VaultFactory.sol: 44~45, 46~47, 48~49	Resolved

Description

The following input parameters for the `mint` function is missing NatSpec comments.

- `_usdc`
- `_supply`
- `_fee`

Recommendation

Consider adding NatSpec comments for improved clarity.

Alleviation

[Co-Museum]: The team resolved this issue in commit 023de14ab57cdbd5479505ce9c07a0e39d9e1824 by adding NatSpec comments.

NFT-01 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Coding Style	Informational	nfts/ERC721ArtNFT.sol: 42; nfts/ERC721HonoraryMembership.sol: 43; nfts/ERC721MembershipUpgradeable.sol: 251	Resolved

Description

The following functions that affect the status of sensitive variables should emit events.

- `setBaseURI()`

Recommendation

Consider adding events for sensitive actions and emit them in the function.

Alleviation

[Co-Museum]: The team resolved this issue in commit 9528d93763b322212b2340139a6093a71993a404 by adding the events.

SET-02 | LACK OF INPUT VALIDATION

Category	Severity	Location	Status
Coding Style	Informational	fractional/Settings.sol: 105~106	Resolved

Description

There is no maximum threshold for the maxCuratorFee which could result in curator fee rates that are greater than 100%.

Recommendation

Consider adding an input validation if there should be a maximum curator rate.

Alleviation

[Co-Museum]: The team resolved this issue in commit 5df1c975a218c1fab8db50d05f7af43f7b1ead5d by adding the check.

OPTIMIZATIONS | CO-MUSEUM

ID	Title	Category	Severity	Status
ACB-07	Variables That Could Be Declared As Immutable	Gas Optimization	Optimization	Resolved
ACB-08	Missing Break In Loop	Gas Optimization	Optimization	Resolved
CON-02	Unused Import File	Coding Style	Optimization	Resolved
ERT-10	Lack Of Validation	Gas Optimization	Optimization	Resolved

ACB-07 | VARIABLES THAT COULD BE DECLARED AS IMMUTABLE

Category	Severity	Location	Status
Gas Optimization	Optimization	crowdsale/AllowanceCrowdsale.sol: 53, 56, 60, 62	Resolved

Description

The linked variables assigned in the constructor can be declared as `immutable`. Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. A big advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since they will not be stored in storage.

Recommendation

Consider declaring these variables as immutable for improved gas cost.

Alleviation

[Co-Museum]: The team resolved this issue in commit dce63c69e95ffabf0f5dbb7a94911859230e1113 by declaring these variables as immutable.

ACB-08 | MISSING BREAK IN LOOP

Category	Severity	Location	Status
Gas Optimization	Optimization	crowdsale/AllowanceCrowdsale.sol: 215~219	Resolved

Description

```
1      for (uint256 i = 0; i < acceptedStablecoins.length; i++) {
2              if (stablecoinAddress == acceptedStablecoins[i]) {
3                  hasTokenAddress = true;
4              }
5      }
```

If the `stablecoinAddress` is found, the loop can finish immediately to save gas.

Recommendation

Add a `break` if the `stablecoinAddress` is found:

```
for (uint256 i = 0; i < acceptedStablecoins.length; i++) {
    if (stablecoinAddress == acceptedStablecoins[i]) {
        hasTokenAddress = true;
        break;
    }
}
```

Alleviation

[Co-Museum]: The team resolved this issue in commit 6b6ff70e2fa148bc52c6a86b25b03c3842119e43 by adding `break`.

CON-02 | UNUSED IMPORT FILE

Category	Severity	Location	Status
Coding Style	Optimization	crowdsale/AllowanceCrowdsale.sol: 5, 7; fractional/ERC721VaultFactory.sol: 7~8, 8~9, 11~12; nfts/ERC721MembershipUpgradeable.sol: 5~6, 8~9	Resolved

Description

In `AllowanceCrowdsale.sol`, the following files are imported but unused.

- `SafeMath.sol`
- `Math.sol`

In `ERC721VaultFactory.sol`, the following files are imported but unused.

- `Settings.sol`
- `ERC721Holder.sol`

In `ERC721MembershipUpgradeable.sol`, the following files are imported but unused.

- `IAccessControlUpgradeable.sol`
- `IERC20Metadata.sol`

Recommendation

Consider removing unnecessary imports for improved legibility and clarity.

Alleviation

[Co-Museum]: The team resolved this issue in commit 411891450307de15e43a257bd79a6cb5b093c389 by removing unnecessary imports.

ERT-10 | LACK OF VALIDATION

Category	Severity	Location	Status
Gas Optimization	Optimization	fractional/ERC721TokenVault.sol: 250~253	Resolved

Description

Throughout the operation of the function `_claimFees()`, there is no check whether `feePerSecond` is greater than 0. If `feePerSecond` is greater than 0, the computations would be unnecessary and only introduce higher gas cost.

Recommendation

Consider adding `if (feePerSecond > 0)` in order to perform computations only when there are fees to be collected.

Alleviation

[Co-Museum]: The team resolved this issue in commit 5da720271a5bdc0ae5727935c6c4877656bba15d by adding `if (feePerSecond > 0)`.

APPENDIX | CO-MUSEUM

Finding Categories

Categories	Description
Centralization / Privilege	Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Logical Issue	Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.
Control Flow	Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.
Language Specific	Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.
Coding Style	Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.
Inconsistency	Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

DISCLAIMER | CERTIK

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 Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. 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 CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. 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.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE

FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

CertiK | Securing the Web3 World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.