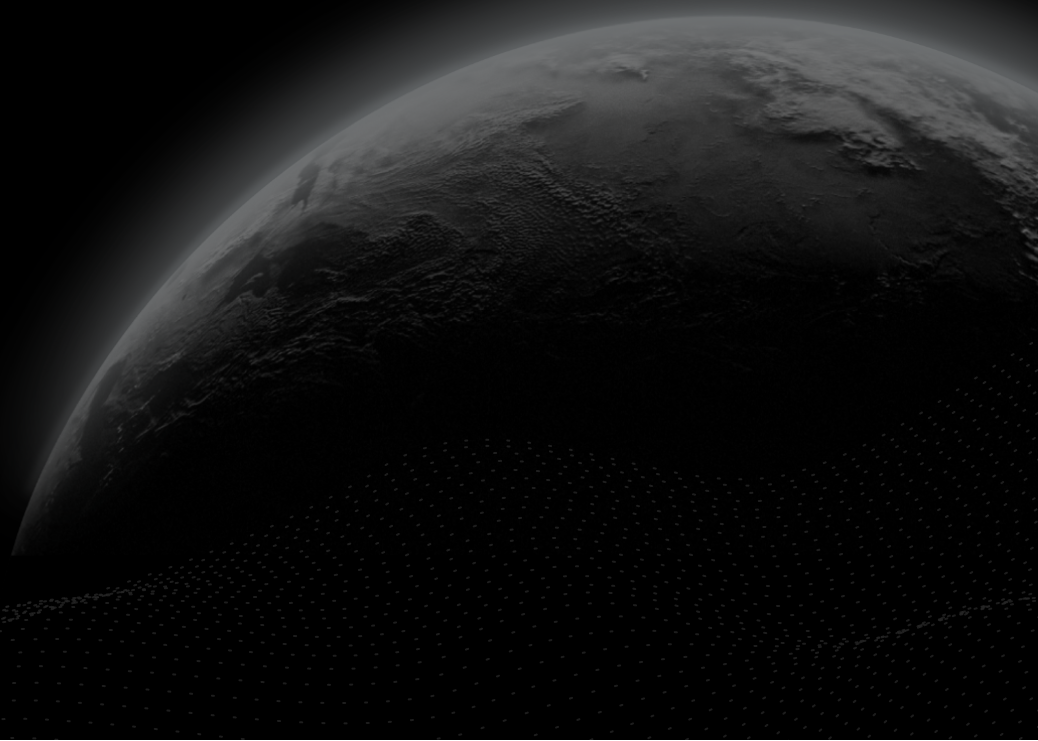
CERTIK

Security Assessment

Crypto Chronic

CertiK Assessed on Jun 29th, 2023



CertiK Assessed on Jun 29th, 2023

Crypto Chronic

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

Executive Summary

TYPES	ECOSYSTEM	METHODS
DeFi	Ethereum (ETH)	Formal Verification, Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 06/29/2023	N/A

CODEBASE

Breeder.sol, Chronic.sol, Feeder.sol

View All in Codebase Page

COMMITS

MD5 (Breeder.sol) = e76f584e770977ee55dc38ed16d9d9fe,
MD5 (Chronic.sol) = 264563bd10ad22f47985ae519f13fb2c,
MD5 (Feeder.sol) = 200fcf011ad7e8cbda414128e293bada

View All in Codebase Page

Vulnerability Summary

8	2	0	0	6	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined

0	Critical		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.
1	Major	1 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.
0	Medium		Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
3	Minor	1 Resolved, 2 Acknowledged	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.
4	Informational	1 Resolved, 3 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 | CRYPTO CHRONIC

CODEBASE | CRYPTO CHRONIC

Repository

Breeder.sol, Chronic.sol, Feeder.sol

Commit

MD5 (Breeder.sol) = e76f584e770977ee55dc38ed16d9d9fe,
MD5 (Chronic.sol) = 264563bd10ad22f47985ae519f13fb2c,
MD5 (Feeder.sol) = 200fcf011ad7e8cbda414128e293bada

AUDIT SCOPE | CRYPTO CHRONIC

3 files audited ● 3 files with Acknowledged findings

ID	File	SHA256 Checksum
● BCC	Breeder.sol	dcb67a400cdc20146261e52b56d208fd491ce1c0fd16172416f46f786ce51bbf
● CCP	Chronic.sol	d312a66c7c5347e12e1a45adef18cc307a6052b080c80dd5d976d8bb4d3fac3f
● FCC	Feeder.sol	e4e38ce31958e2bd8586725eb8cd90325212846ac98b76f0a99bde3e6bc6995a

APPROACH & METHODS | CRYPTO CHRONIC

This report has been prepared for Crypto Chronic to discover issues and vulnerabilities in the source code of the Crypto Chronic 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 critical 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:

- Testing the smart contracts against both common and uncommon attack vectors;
- 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.

FINDINGS | CRYPTO CHRONIC

8	0	1	0	3	4
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Crypto Chronic. Through this audit, we have uncovered 8 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:

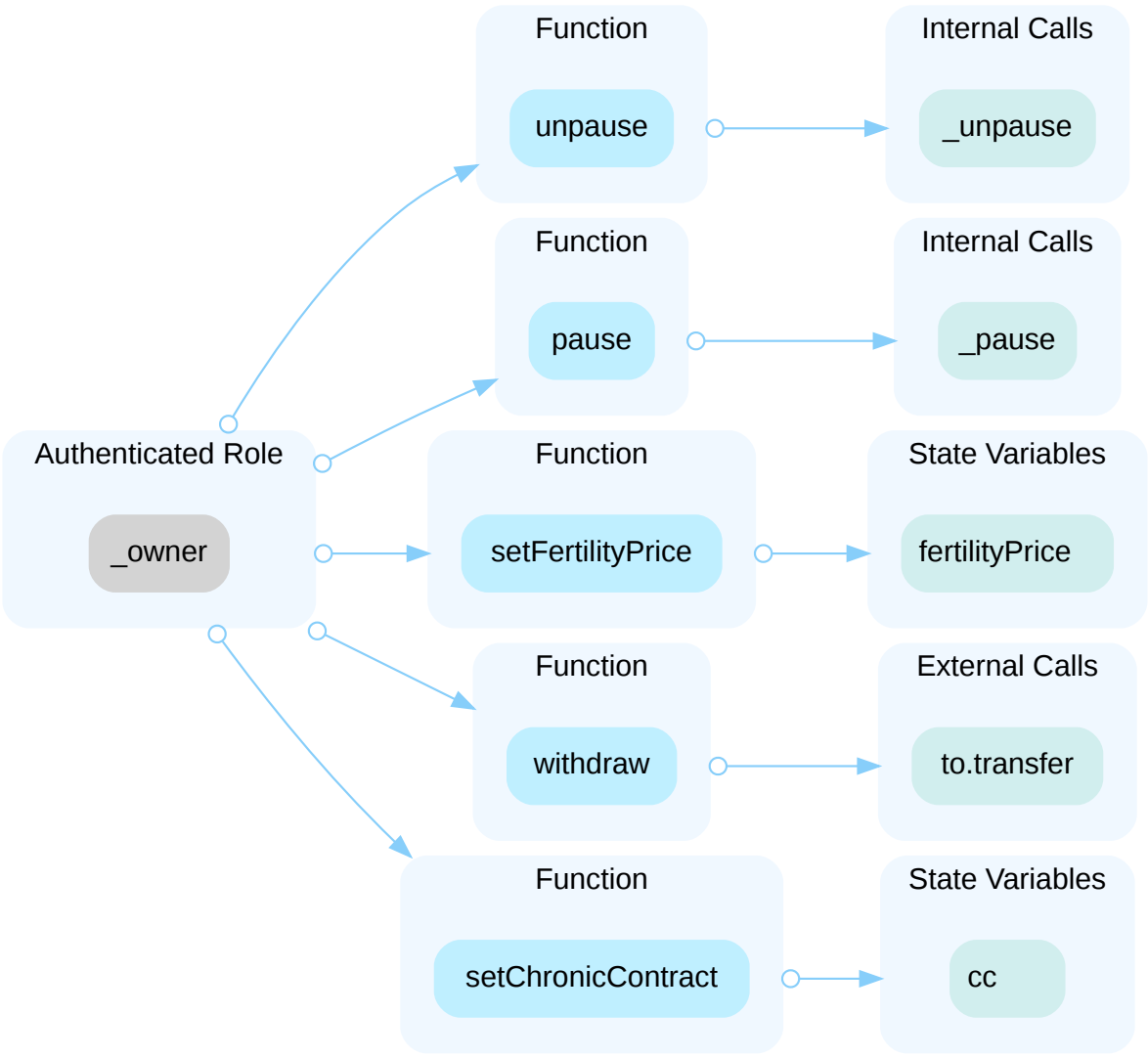
ID	Title	Category	Severity	Status
CCK-01	**Centralization Related Risks**	**Centralization**	**Major**	**Acknowledged**
BCC-01	Unused Return Value	Volatile Code	Minor	Acknowledged
BCC-02	Weak PRNG	Volatile Code	Minor	Acknowledged
CCK-03	Usage Of `transfer` / `send` For Sending Ether	Volatile Code	Minor	Resolved
BCC-03	Reusable `signature`	Logical Issue	Informational	Acknowledged
CCK-04	Missing Error Messages	Coding Style	Informational	Resolved
CCK-05	Missing Emit Events	Coding Style	Informational	Acknowledged
CCP-01	Missing Zero Address Validation	Volatile Code	Informational	Acknowledged

CCK-01 | CENTRALIZATION RELATED RISKS

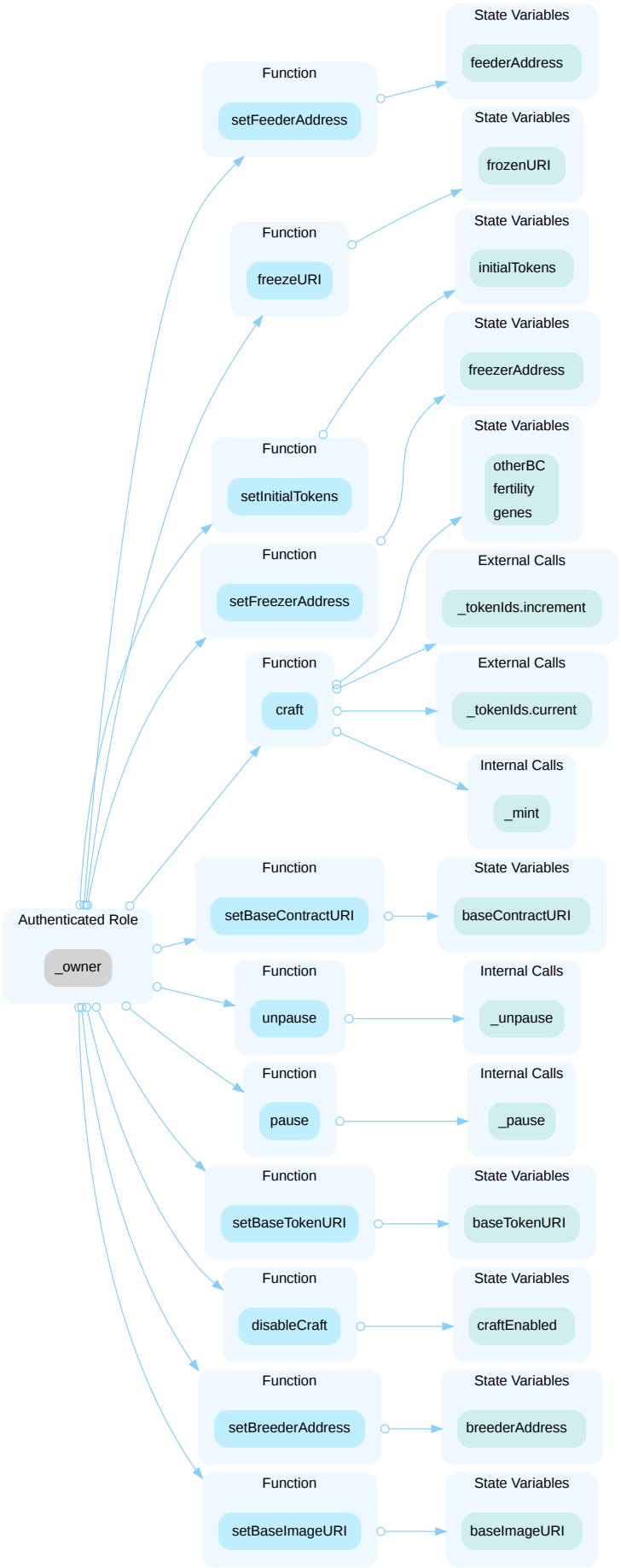
Category	Severity	Location	Status
Centralization	Major	Breeder.sol: 520, 524, 556, 560, 566, 572, 578, 584, 588, 596, 606, 615, 661, 682; Chronic.sol: 1941, 1947, 1953, 1957, 1962, 1974, 1986, 1997, 2003, 2015, 2027, 2040, 2111, 2116, 2126, 2137; Feeder.sol: 442, 446, 451, 455, 479	Acknowledged

Description

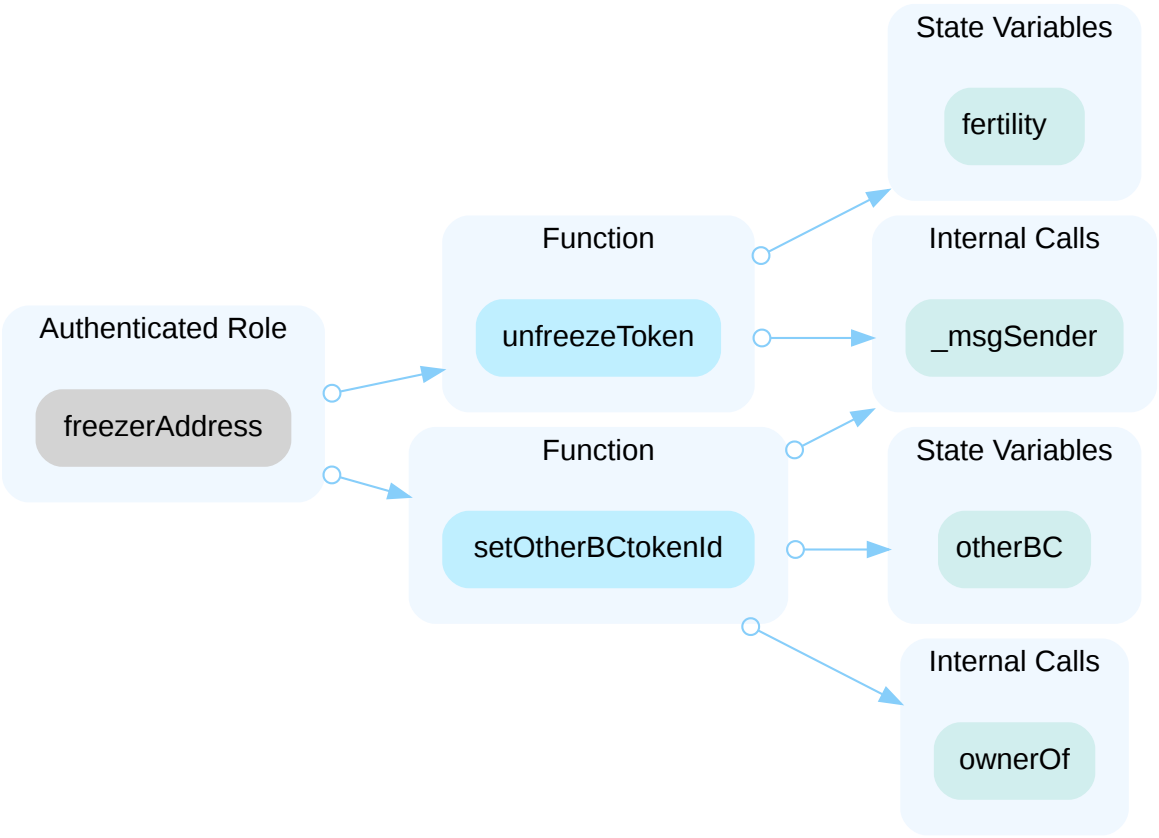
In the contract `Feeder` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority to pause/resume functionalities of this contract, set the `Chronic` contract address, set the fertility price, withdraw ETH from the contract, renounce ownership, and transfer ownership to a new owner.



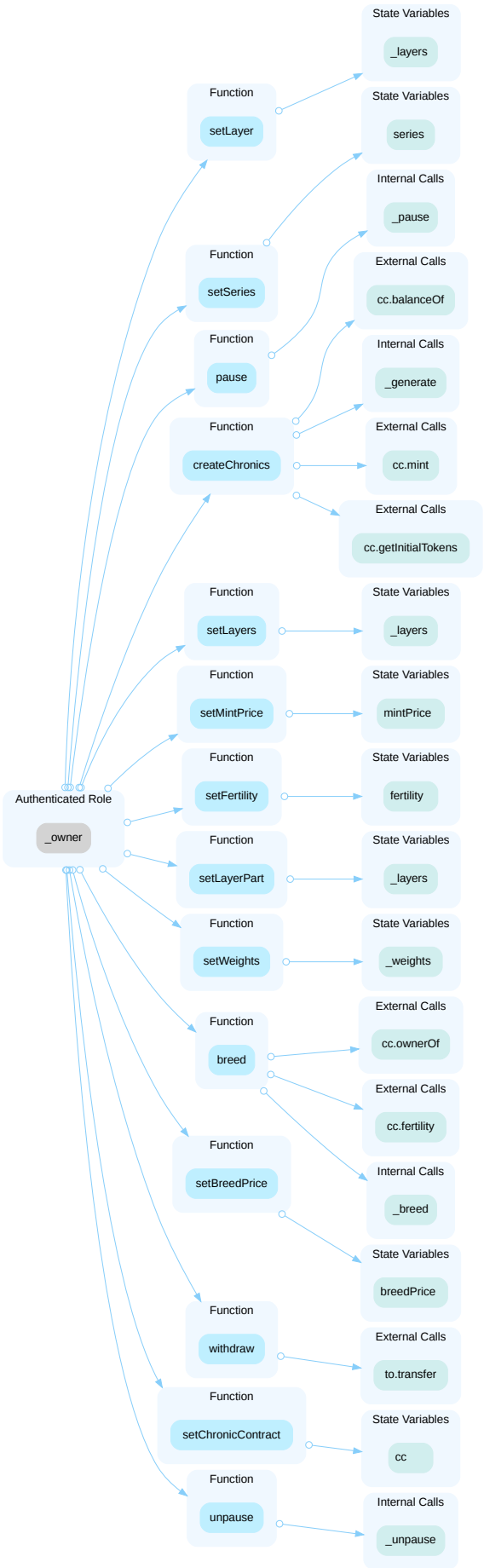
In the contract `Chronic` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and disable the craft, freeze URI, pause/resume the contract, set the `Breeder` contract address, set the `Feeder` contract address, set the freezer address, set the initial token value, set the base token URI, set the base image URI, set the base contract URI, craft NFT to anyone, renounce ownership, and transfer ownership to a new owner.


Authenticated Role
_owner
Function
setFeederAddress
State Variables
feederAddress
Function
freezeURI
State Variables
frozenURI
Function
setInitialTokens
State Variables
initialTokens
State Variables
freezerAddress
State Variables
otherBC
fertility
genes
Function
setFreezerAddress
External Calls
_tokenIds.increment
Function
craft
External Calls
_tokenIds.current
Internal Calls
_mint
Function
setBaseContractURI
State Variables
baseContractURI
Function
unpause
Internal Calls
_unpause
Function
pause
Internal Calls
_pause
Function
setBaseTokenURI
State Variables
baseTokenURI
Function
disableCraft
State Variables
craftEnabled
Function
setBreederAddress
State Variables
breederAddress
Function
setBaseImageURI
State Variables
baseImageURI

In the contract `Chronic` the role `freezerAddress` has authority over the functions shown in the diagram below. Any compromise to the `freezerAddress` account may allow the hacker to take advantage of this authority to `unfreezeToken` and `setOtherBCtokenId`.



In the contract `Breeder` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and pause/resume the contract, set the `Chronic` contract address, set the breed price, set mint price, set fertility, set series, set layers, set layer value, set layer part, set weights, create CryptoChronicNFT, breed the gene, withdraw ETH from the contract, renounce ownership, and transfer ownership to a new owner.


Authenticated Role
_owner
Function
setLayer
State Variables
_layers
Function
setSeries
State Variables
series
Function
pause
Internal Calls
_pause
Function
createChronics
External Calls
cc.balanceOf
Internal Calls
_generate
External Calls
cc.mint
External Calls
cc.getInitialTokens
Function
setLayers
State Variables
_layers
Function
setMintPrice
State Variables
mintPrice
Function
setFertility
State Variables
fertility
Function
setLayerPart
State Variables
_layers
Function
setWeights
State Variables
_weights
Function
breed
External Calls
cc.ownerOf
External Calls
cc.fertility
Internal Calls
_breed
Function
setBreedPrice
State Variables
breedPrice
Function
withdraw
External Calls
to.transfer
Function
setChronicContract
State Variables
cc
Function
unpause
Internal Calls
_unpause

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

[Crypto Chronic Team]:

We have improved this issue, albeit to some extent it is congenital to our model. In order to alleviate it, we have implemented access control with 3 roles and 1 owner. Only the Owner role can set the access control, while only the Finance role can retrieve ETH from the contract, and only the Admin role can do the setup required by the game. Finally, only the Whitelister role can setup whitelisting for promo (both now and in the future). Although we commit to further alleviating the issue as soon as possible, for example by implementing multi-signature wallets and a time-lock with reasonable latency (48H) for awareness on privileged operations, it cannot be completely eliminated, given that our design requires these privileges to be in place if we wish to remain true to our vision. Decentralized computing doesn't necessarily need a decentralized market. While the Blockchain is indeed decentralized, major operators and marketplaces are centralized. They, along with us, aim to promote consensus and mass adoption of crypto but must make compromises. We believe that current offerings prioritize decentralization at the expense of democratization due to high costs. Our Unique Value Proposition is to democratize the market through freemium pricing and a free-to-earn model. To achieve this, we must operate on a private chain, sacrificing some decentralization in the pursuit of bringing crypto closer to mass adoption.

BCC-01 | UNUSED RETURN VALUE

Category	Severity	Location	Status
Volatile Code	Minor	Breeder.sol: 621, 636	Acknowledged

Description

The return value of an external call is not stored in a local or state variable.

```
621             cc.mint(to, gene, 0, 0, fertility);
```

```
636         cc.mint(_msgSender(), gene, 0, 0, fertility);
```

Recommendation

We recommend checking or using the return values of all external function calls.

Alleviation

[Crypto Chronic Team]:

As per Solidity requirement, functions that change the Worldstate/Registry should never return a value, as it's not reliable data: the reliable part is in the Event emitted. This value has been returned because of development and backend requirements but is validated using the Event emitted. Hence, there is no need to store the value in a state variable as the value is already stored by the mint function itself. No changes are required, therefore, as Event should be used to acknowledge the return value.

BCC-02 | WEAK PRNG

Category	Severity	Location	Status
Volatile Code	Minor	Breeder.sol: 691	Acknowledged

Description

Weak PRNG due to a modulo on `block.timestamp` and `block.difficulty`. These can be influenced by miners to some extent, so they should be avoided.

```
691         return uint256(keccak256(abi.encodePacked(block.difficulty, block.
timestamp, (_initialNumber++)))) % number;
```

Recommendation

Instead of using `block.timestamp` and `block.difficulty` as a source of randomness, we recommend using a verifiable source of randomness, such as Chainlink VRF(https://docs.chain.link/docs/get-a-random-number/), for the purpose of random number generation.

Alleviation

[Crypto Chronic Team]:

We have alleviated the issue by using PrevranDao instead of Difficulty to improve randomness. In our code there is Difficulty because the Private Chain doesn't normally support PrevranDao.

CCK-03 | USAGE OF `transfer` / `send` FOR SENDING ETHER

Category	Severity	Location	Status
Volatile Code	Minor	Breeder.sol: 685; Feeder.sol: 482	Resolved

Description

It is not recommended to use Solidity's `transfer()` and `send()` functions for transferring Ether, since some contracts may not be able to receive the funds. Those functions forward only a fixed amount of gas (2300 specifically) and the receiving contracts may run out of gas before finishing the transfer. Also, EVM instructions' gas costs may increase in the future. Thus, some contracts that can receive now may stop working in the future due to the gas limitation.

```
685         to.transfer(balance);
```

- `Breeder.withdraw` uses `transfer()`.

```
482         to.transfer(balance);
```

- `Feeder.withdraw` uses `transfer()`.

Recommendation

We recommend using the `Address.sendValue()` function from OpenZeppelin.

Since `Address.sendValue()` may allow reentrancy, we also recommend guarding against reentrancy attacks by utilizing the Checks-Effects-Interactions Pattern or applying OpenZeppelin ReentrancyGuard.

Alleviation

The client revised the code and resolved the issue by using pull payments of openzeppelin that use an escrow mechanism.

BCC-03 | REUSABLE `signature`

Category	Severity	Location	Status
Logical Issue	Informational	Breeder.sol: 2221	Acknowledged

Description

While the `mintWhitelistedChronic()` function requires a `signature` generated by the whitelist, it does not verify if the `signature` has been previously used. Consequently, the `signature` can be reused, enabling anyone to mint `Chronic` with `wlPrice` by utilizing the same `signature` .

Recommendation

We would like to confirm with the client if the current implementation aligns with the original project design.

Alleviation

[Crypto Chronic Team]:

We confirm that the current implementation is deliberate and aligns with the original project design. We will manage the number of mints from the site, but the contract will remain 'free' to be used multiple times within the timeframe in which the Whitelist is active.

CCK-04 | MISSING ERROR MESSAGES

Category	Severity	Location	Status
Coding Style	Informational	Breeder.sol: 515; Feeder.sol: 437	Resolved

Description

The **require** can be used to check for conditions and throw an exception if the condition is not met. It is better to provide a string message containing details about the error that will be passed back to the caller.

Recommendation

We advise adding error messages to the linked **require** statements.

Alleviation

The client revised the code and resolved the issue.

CCK-05 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Coding Style	Informational	Breeder.sol: 556, 584, 588, 606; Feeder.sol: 451	Acknowledged

Description

There should always be events emitted in the sensitive functions that are controlled by centralization roles.

Recommendation

It is recommended emitting events for the sensitive functions that are controlled by centralization roles.

Alleviation

[Crypto Chronic Team]:

We are in the process of reviewing the possibility of emitting events for the sensitive functions that are controlled by centralization roles and will alleviate the issue as soon as we have finalized the best manner to do so.

CCP-01 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	Informational	Chronic.sol: 1934, 1935, 1936	Acknowledged

Description

Addresses should be checked before assignment or external call to make sure they are not zero addresses.

```
1934        breederAddress = _breederAddress;
```

- `_breederAddress` is not zero-checked before being used.

```
1935        feederAddress = _feederAddress;
```

- `_feederAddress` is not zero-checked before being used.

```
1936        freezerAddress = _freezerAddress;
```

- `_freezerAddress` is not zero-checked before being used.

Recommendation

We advise adding a zero-check for the passed-in address value to prevent unexpected errors.

Alleviation

[Crypto Chronic Team]:

Being able to always change these addresses as Owner, we didn't want to implement the suggested checks in order to allow us to 'freeze' the feature during upgrades, assigning it to Address 0. However, we are looking into ways to alleviate the issue finding an efficient and easy to integrate alternative to adding a Zero-Check for the passed-in address value to prevent unexpected errors.

FORMAL VERIFICATION | CRYPTO CHRONIC

Formal guarantees about the behavior of smart contracts can be obtained by reasoning about properties relating to the entire contract (e.g. contract invariants) or to specific functions of the contract. Once such properties are proven to be valid, they guarantee that the contract behaves as specified by the property. As part of this audit, we applied automated formal verification (symbolic model checking) to prove that well-known functions in the smart contracts adhere to their expected behavior.

Considered Functions And Scope

In the following, we provide a description of the properties that have been used in this audit. They are grouped according to the type of contract they apply to.

Verification of ERC-721 Compliance

We verified the properties of the public interface of those token contracts that implement the ERC-721 interface without pause.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
erc721-balanceof-succeed-normal	`balanceOf` Succeeds on Admissible Inputs
erc721-balanceof-correct-count	`balanceOf` Returns the Correct Value
erc721-supportsinterface-correct-erc721	`supportsInterface` Signals Support for `ERC721`
erc721-balanceof-no-change-state	`balanceOf` Does Not Change the Contract's State
erc721-balanceof-revert	`balanceOf` Fails on the Zero Address
erc721-ownerof-succeed-normal	`ownerOf` Succeeds For Valid Tokens
erc721-ownerof-correct-owner	`ownerOf` Returns the Correct Owner
erc721-ownerof-revert	`ownerOf` Fails On Invalid Tokens
erc721-ownerof-no-change-state	`ownerOf` Does Not Change the Contract's State
erc721-transferfrom-succeed-normal	`transferFrom` Succeeds on Admissible Inputs
erc721-getapproved-correct-value	`getApproved` Returns Correct Approved Address
erc721-getapproved-succeed-normal	`getApproved` Succeeds For Valid Tokens
erc721-getapproved-revert-zero	`getApproved` Fails on Invalid Tokens

Property Name	Title
erc721-getapproved-change-state	`getApproved` Does Not Change the Contract's State
erc721-isapprovedforall-succeed-normal	`isApprovedForAll` Always Succeeds
erc721-isapprovedforall-correct	`isApprovedForAll` Returns Correct Approvals
erc721-isapprovedforall-change-state	`isApprovedForAll` Does Not Change the Contract's State
erc721-approve-succeed-normal	`approve` Returns for Admissible Inputs
erc721-approve-set-correct	`approve` Sets Approval
erc721-approve-revert-not-allowed	`approve` Prevents Unpermitted Approvals
erc721-approve-revert-invalid-token	`approve` Fails For Calls with Invalid Tokens
erc721-setapprovalforall-succeed-normal	`setApprovalForAll` Returns for Admissible Inputs
erc721-approve-change-state	`approve` Has No Unexpected State Changes
erc721-setapprovalforall-multiple	`setApprovalForAll` Can Set Multiple Operators
erc721-setapprovalforall-set-correct	`setApprovalForAll` Approves Operator
erc721-setapprovalforall-change-state	`setApprovalForAll` Has No Unexpected State Changes
erc721-transferfrom-correct-one-token-self	`transferFrom` Performs Self Transfers Correctly
erc721-transferfrom-correct-approval	`transferFrom` Updates the Approval Correctly
erc721-transferfrom-correct-increase	`transferFrom` Transfers the Complete Token in Non-self Transfers
erc721-transferfrom-correct-owner-from	`transferFrom` Removes Token Ownership of From
erc721-transferfrom-correct-owner-to	`transferFrom` Transfers Ownership
erc721-transferfrom-correct-balance	`transferFrom` Sum of Balances is Constant
erc721-transferfrom-correct-state-balance	`transferFrom` Keeps Balances Constant Except for From and To
erc721-transferfrom-correct-state-owner	`transferFrom` Has Expected Ownership Changes
erc721-transferfrom-correct-state-approval	`transferFrom` Has Expected Approval Changes
erc721-transferfrom-revert-invalid	`transferFrom` Fails for Invalid Tokens
erc721-transferfrom-revert-from-zero	`transferFrom` Fails for Transfers From the Zero Address

Property Name	Title
erc721-transferfrom-revert-to-zero	`transferFrom` Fails for Transfers To the Zero Address
erc721-supportsinterface-metadata	`supportsInterface` Signals that ERC721Metadata is Implemented
erc721-supportsinterface-succeed-always	`supportsInterface` Always Succeeds
erc721-supportsinterface-correct-erc165	`supportsInterface` Signals Support for ERC165
erc721-supportsinterface-correct-false	`supportsInterface` Returns `False` for Id 0xffffffff
erc721-supportsinterface-no-change-state	`supportsInterface` Does Not Change the Contract's State
erc721-transferfrom-revert-not-owned	`transferFrom` Fails if `From` Is Not Token Owner
erc721-transferfrom-revert-exceed-approval	`transferFrom` Fails for Token Transfers without Approval

Verification of Compliance with Pausable ERC-721

We verified the properties of the public interface of those token contracts that implement the pausable ERC-721 interface.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
erc721pausable-transferfrom-succeed-normal	`transferFrom` Succeeds on Admissible Inputs
erc721pausable-transferfrom-revert-pause	`transferFrom` Fails when Paused
erc721pausable-supportsinterface-correct-erc721	`supportsInterface` Signals Support for `ERC721`
erc721pausable-balanceof-succeed-normal	`balanceOf` Succeeds on Admissible Inputs
erc721pausable-balanceof-correct-count	`balanceOf` Returns the Correct Value
erc721pausable-balanceof-no-change-state	`balanceOf` Does Not Change the Contract's State
erc721pausable-balanceof-revert	`balanceOf` Fails on the Zero Address
erc721pausable-ownerof-succeed-normal	`ownerOf` Succeeds For Valid Tokens
erc721pausable-ownerof-correct-owner	`ownerOf` Returns the Correct Owner
erc721pausable-ownerof-no-change-state	`ownerOf` Does Not Change the Contract's State
erc721pausable-ownerof-revert	`ownerOf` Fails On Invalid Tokens

Property Name	Title
erc721pausable-getapproved-change-state	`getApproved` Does Not Change the Contract's State
erc721pausable-getapproved-succeed-normal	`getApproved` Succeeds For Valid Tokens
erc721pausable-getapproved-correct-value	`getApproved` Returns Correct Approved Address
erc721pausable-getapproved-revert-zero	`getApproved` Fails on Invalid Tokens
erc721pausable-isapprovedforall-change-state	`isApprovedForAll` Does Not Change the Contract's State
erc721pausable-isapprovedforall-succeed-normal	`isApprovedForAll` Always Succeeds
erc721pausable-isapprovedforall-correct	`isApprovedForAll` Returns Correct Approvals
erc721pausable-approve-succeed-normal	`approve` Returns for Admissible Inputs
erc721pausable-approve-set-correct	`approve` Sets Approval
erc721pausable-approve-revert-invalid-token	`approve` Fails For Calls with Invalid Tokens
erc721pausable-approve-revert-not-allowed	`approve` Prevents Unpermitted Approvals
erc721pausable-approve-change-state	`approve` Has No Unexpected State Changes
erc721pausable-setapprovalforall-succeed-normal	`setApprovalForAll` Returns for Admissible Inputs
erc721pausable-setapprovalforall-change-state	`setApprovalForAll` Has No Unexpected State Changes
erc721pausable-setapprovalforall-set-correct	`setApprovalForAll` Approves Operator
erc721pausable-setapprovalforall-multiple	`setApprovalForAll` Can Set Multiple Operators
erc721pausable-transferfrom-correct-increase	`transferFrom` Transfers the Complete Token in Non-self Transfers
erc721pausable-transferfrom-correct-one-token-self	`transferFrom` Performs Self Transfers Correctly
erc721pausable-transferfrom-correct-approval	`transferFrom` Updates the Approval Correctly
erc721pausable-transferfrom-correct-owner-from	`transferFrom` Removes Token Ownership of From
erc721pausable-transferfrom-correct-owner-to	`transferFrom` Transfers Ownership
erc721pausable-transferfrom-correct-balance	`transferFrom` Sum of Balances is Constant

Property Name	Title
erc721pausable-transferfrom-correct-state-balance	`transferFrom` Keeps Balances Constant Except for From and To
erc721pausable-transferfrom-correct-state-owner	`transferFrom` Has Expected Ownership Changes
erc721pausable-transferfrom-correct-state-approval	`transferFrom` Has Expected Approval Changes
erc721pausable-transferfrom-revert-invalid	`transferFrom` Fails for Invalid Tokens
erc721pausable-transferfrom-revert-from-zero	`transferFrom` Fails for Transfers From the Zero Address
erc721pausable-transferfrom-revert-to-zero	`transferFrom` Fails for Transfers To the Zero Address
erc721pausable-transferfrom-revert-not-owned	`transferFrom` Fails if `From` Is Not Token Owner
erc721pausable-transferfrom-revert-exceed-approval	`transferFrom` Fails for Token Transfers without Approval
erc721pausable-totalsupply-change-state	`totalSupply` Does Not Change the Contract's State
erc721pausable-supportsinterface-metadata	`supportsInterface` Signals that ERC721Metadata is Implemented
erc721pausable-totalsupply-succeed-always	`totalSupply` Always Succeeds
erc721pausable-supportsinterface-enumerable	`supportsInterface` Signals that ERC721Enumerable is Implemented
erc721pausable-supportsinterface-succeed-always	`supportsInterface` Always Succeeds
erc721pausable-tokenofownerbyindex-revert	`tokenOfOwnerByIndex` Correctly Fails on Token Owner Indices Greater as the Owner Balance
erc721pausable-supportsinterface-correct-erc165	`supportsInterface` Signals Support for ERC165
erc721pausable-supportsinterface-correct-false	`supportsInterface` Returns `False` for Id 0xffffffff
erc721pausable-supportsinterface-no-change-state	`supportsInterface` Does Not Change the Contract's State

Verification Results

For the following contracts, model checking established that each of the properties that were in scope of this audit (see scope) are valid:

Detailed Results For Contract ERC721 (projects/CryptoChronic/contracts/Chronic.sol) In Commit 5de987ed3ed354c91e2ed01682aab95c417f7d7c

Verification of ERC-721 Compliance

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc721-balanceof-succeed-normal	True	
erc721-balanceof-correct-count	True	
erc721-balanceof-no-change-state	True	
erc721-balanceof-revert	True	

Detailed results for function `supportsInterface`

Property Name	Final Result	Remarks
erc721-supportsinterface-correct-erc721	True	
erc721-supportsinterface-metadata	True	
erc721-supportsinterface-succeed-always	True	
erc721-supportsinterface-correct-erc165	True	
erc721-supportsinterface-correct-false	True	
erc721-supportsinterface-no-change-state	True	

Detailed results for function `ownerOf`

Property Name	Final Result	Remarks
erc721-ownerof-succeed-normal	True	
erc721-ownerof-correct-owner	True	
erc721-ownerof-revert	True	
erc721-ownerof-no-change-state	True	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc721-transferfrom-succeed-normal	True	
erc721-transferfrom-correct-one-token-self	True	
erc721-transferfrom-correct-approval	True	
erc721-transferfrom-correct-increase	True	
erc721-transferfrom-correct-owner-from	True	
erc721-transferfrom-correct-owner-to	True	
erc721-transferfrom-correct-balance	True	
erc721-transferfrom-correct-state-balance	True	
erc721-transferfrom-correct-state-owner	True	
erc721-transferfrom-correct-state-approval	True	
erc721-transferfrom-revert-invalid	True	
erc721-transferfrom-revert-from-zero	True	
erc721-transferfrom-revert-to-zero	True	
erc721-transferfrom-revert-not-owned	True	
erc721-transferfrom-revert-exceed-approval	True	

Detailed results for function `getApproved`

Property Name	Final Result	Remarks
erc721-getapproved-correct-value	True	
erc721-getapproved-succeed-normal	True	
erc721-getapproved-revert-zero	True	
erc721-getapproved-change-state	True	

Detailed results for function `isApprovedForAll`

Property Name	Final Result	Remarks
erc721-isapprovedforall-succeed-normal	True	
erc721-isapprovedforall-correct	True	
erc721-isapprovedforall-change-state	True	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc721-approve-succeed-normal	True	
erc721-approve-set-correct	True	
erc721-approve-revert-not-allowed	True	
erc721-approve-revert-invalid-token	True	
erc721-approve-change-state	True	

Detailed results for function `setApprovalForAll`

Property Name	Final Result	Remarks
erc721-setapprovalforall-succeed-normal	True	
erc721-setapprovalforall-multiple	True	
erc721-setapprovalforall-set-correct	True	
erc721-setapprovalforall-change-state	True	

In the remainder of this section, we list all contracts where model checking of at least one property was not successful. There are several reasons why this could happen:

- Model checking reports a counterexample that violates the property. Depending on the counterexample,this occurs if
    - The specification of the property is too generic and does not accurately capture the intended behavior of the smart contract. In that case, the counterexample does not indicate a problem in the underlying smart contract. We report such instances as being "inapplicable".
    - The property is applicable to the smart contract. In that case, the counterexample showcases a problem in the smart contract and a correspond finding is reported separately in the Findings section of this

report. In the following tables, we report such instances as "invalid". The distinction between spurious and actual counterexamples is done manually by the auditors.

- The model checking result is inconclusive. Such a result does not indicate a problem in the underlying smart contract. An inconclusive result may occur if
    - The model checking engine fails to construct a proof. This can happen if the logical deductions necessary are beyond the capabilities of the automated reasoning tool. It is a technical limitation of all proof engines and cannot be avoided in general.
    - The model checking engine runs out of time or memory and did not produce a result. This can happen if automatic abstraction techniques are ineffective or of the state space is too big.

Detailed Results For Contract Chronic (projects/CryptoChronic/contracts/Chronic.sol) In Commit 5de987ed3ed354c91e2ed01682aab95c417f7d7c

Verification of Compliance with Pausable ERC-721

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc721pausable-transferfrom-succeed-normal	Inconclusive	
erc721pausable-transferfrom-revert-pause	Inconclusive	
erc721pausable-transferfrom-correct-increase	Inconclusive	
erc721pausable-transferfrom-correct-one-token-self	Inconclusive	
erc721pausable-transferfrom-correct-approval	Inconclusive	
erc721pausable-transferfrom-correct-owner-from	Inconclusive	
erc721pausable-transferfrom-correct-owner-to	Inconclusive	
erc721pausable-transferfrom-correct-balance	Inconclusive	
erc721pausable-transferfrom-correct-state-balance	Inconclusive	
erc721pausable-transferfrom-correct-state-owner	Inconclusive	
erc721pausable-transferfrom-correct-state-approval	Inconclusive	
erc721pausable-transferfrom-revert-invalid	Inconclusive	
erc721pausable-transferfrom-revert-from-zero	Inconclusive	
erc721pausable-transferfrom-revert-to-zero	Inconclusive	
erc721pausable-transferfrom-revert-not-owned	Inconclusive	
erc721pausable-transferfrom-revert-exceed-approval	Inconclusive	

Detailed results for function `supportsInterface`

Property Name	Final Result	Remarks
erc721pausable-supportsinterface-correct-erc721	True	
erc721pausable-supportsinterface-metadata	True	
erc721pausable-supportsinterface-enumerable	True	
erc721pausable-supportsinterface-succeed-always	True	
erc721pausable-supportsinterface-correct-erc165	True	
erc721pausable-supportsinterface-correct-false	True	
erc721pausable-supportsinterface-no-change-state	Inconclusive	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc721pausable-balanceof-succeed-normal	True	
erc721pausable-balanceof-correct-count	True	
erc721pausable-balanceof-no-change-state	Inconclusive	
erc721pausable-balanceof-revert	True	

Detailed results for function `ownerOf`

Property Name	Final Result	Remarks
erc721pausable-ownerof-succeed-normal	True	
erc721pausable-ownerof-correct-owner	True	
erc721pausable-ownerof-no-change-state	Inconclusive	
erc721pausable-ownerof-revert	True	

Detailed results for function `getApproved`

Property Name	Final Result	Remarks
erc721pausable-getapproved-change-state	Inconclusive	
erc721pausable-getapproved-succeed-normal	True	
erc721pausable-getapproved-correct-value	True	
erc721pausable-getapproved-revert-zero	True	

Detailed results for function `isApprovedForAll`

Property Name	Final Result	Remarks
erc721pausable-isapprovedforall-change-state	Inconclusive	
erc721pausable-isapprovedforall-succeed-normal	True	
erc721pausable-isapprovedforall-correct	True	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc721pausable-approve-succeed-normal	True	
erc721pausable-approve-set-correct	True	
erc721pausable-approve-revert-invalid-token	True	
erc721pausable-approve-revert-not-allowed	True	
erc721pausable-approve-change-state	Inconclusive	

Detailed results for function `setApprovalForAll`

Property Name	Final Result	Remarks
erc721pausable-setapprovalforall-succeed-normal	True	
erc721pausable-setapprovalforall-change-state	Inconclusive	
erc721pausable-setapprovalforall-set-correct	True	
erc721pausable-setapprovalforall-multiple	True	

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc721pausable-totalsupply-change-state	Inconclusive	
erc721pausable-totalsupply-succeed-always	True	

Detailed results for function `tokenOfOwnerByIndex`

Property Name	Final Result	Remarks
erc721pausable-tokenofownerbyindex-revert	True	

APPENDIX | CRYPTO CHRONIC

Finding Categories

Categories	Description
Centralization	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.
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.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.
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.

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.

Details on Formal Verification

Technical description

Some Solidity smart contracts from this project have been formally verified using symbolic model checking. Each such contract was compiled into a mathematical model which reflects all its possible behaviors with respect to the property. The model takes into account the semantics of the Solidity instructions found in the contract. All verification results that we report are based on that model.

The model also formalizes a simplified execution environment of the Ethereum blockchain and a verification harness that performs the initialization of the contract and all possible interactions with the contract. Initially, the contract state is initialized non-deterministically (i.e. by arbitrary values) and over-approximates the reachable state space of the contract throughout any actual deployment on chain. All valid results thus carry over to the contract's behavior in arbitrary states after it has been deployed.

Assumptions and simplifications

The following assumptions and simplifications apply to our model:

- Gas consumption is not taken into account, i.e. we assume that executions do not terminate prematurely because they run out of gas.
- The contract's state variables are non-deterministically initialized before invocation of any of those functions. That ignores contract invariants and may lead to false positives. It is, however, a safe over-approximation.
- The verification engine reasons about unbounded integers. Machine arithmetic is modeled as operations on the congruence classes arising from the bit-width of the underlying numeric type. This ensures that over- and underflow characteristics are faithfully represented.
- Certain low-level calls and inline assembly are not supported and may lead to an ERC-20 token contract not being formally verified.
- We model the semantics of the Solidity source code and not the semantics of the EVM bytecode in a compiled contract.

Formalism for property definitions

All properties are expressed in linear temporal logic (LTL). For that matter, we treat each invocation of and each return from a public or an external function as a discrete time steps. Our analysis reasons about the contract's state upon entering and upon leaving public or external functions.

Apart from the Boolean connectives and the modal operators "always" (written `[]`) and "eventually" (written `<>`), we use the following predicates to reason about the validity of atomic propositions. They are evaluated on the contract's state whenever a discrete time step occurs:

- `started(f, [cond])` Indicates an invocation of contract function `f` within a state satisfying formula `cond`.
- `willSucceed(f, [cond])` Indicates an invocation of contract function `f` within a state satisfying formula `cond` and considers only those executions that do not revert.
- `finished(f, [cond])` Indicates that execution returns from contract function `f` in a state satisfying formula `cond`. Here, formula `cond` may refer to the contract's state variables and to the value they had upon entering the function (using the `old` function).
- `reverted(f, [cond])` Indicates that execution of contract function `f` was interrupted by an exception in a contract state satisfying formula `cond`.

The verification performed in this audit operates on a harness that non-deterministically invokes a function of the contract's public or external interface. All formulas are analyzed w.r.t. the trace that corresponds to this function invocation.

Description of ERC-20 Properties

The specifications are designed such that they capture the desired and admissible behaviors of the ERC-20 functions `transfer`, `transferFrom`, `approve`, `allowance`, `balanceOf`, and `totalSupply`.

In the following, we list those property specifications.

Properties for ERC-20 function `transfer`

erc20-transfer-revert-zero

Function `transfer` Prevents Transfers to the Zero Address.

Any call of the form `transfer(recipient, amount)` must fail if the recipient address is the zero address.

Specification:

```
[](started(contract.transfer(to, value), to == address(0))
   ==> <>(reverted(contract.transfer) || finished(contract.transfer(to, value),
          !return)))
```

erc20-transfer-succeed-normal

Function `transfer` Succeeds on Admissible Non-self Transfers.

All invocations of the form `transfer(recipient, amount)` must succeed and return `true` if

- the `recipient` address is not the zero address,
- `amount` does not exceed the balance of address `msg.sender`,
- transferring `amount` to the `recipient` address does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call.

Specification:

```
[](started(contract.transfer(to, value), to != address(0)
          && to != msg.sender && value >= 0 && value <= _balances[msg.sender]
          && _balances[to] + value <= type(uint256).max && _balances[to] >= 0
          && _balances[msg.sender] <= type(uint256).max)
          ==> <>(finished(contract.transfer(to, value), return)))
```

erc20-transfer-succeed-self

Function `transfer` Succeeds on Admissible Self Transfers.

All self-transfers, i.e. invocations of the form `transfer(recipient, amount)` where the `recipient` address equals the address in `msg.sender` must succeed and return `true` if

- the value in `amount` does not exceed the balance of `msg.sender` and
- the supplied gas suffices to complete the call.

Specification:

```
[](started(contract.transfer(to, value), to != address(0)
          && to == msg.sender && value >= 0 && value <= _balances[msg.sender]
          && _balances[msg.sender] >= 0
          && _balances[msg.sender] <= type(uint256).max)
          ==> <>(finished(contract.transfer(to, value), return)))
```

erc20-transfer-correct-amount

Function `transfer` Transfers the Correct Amount in Non-self Transfers.

All non-reverting invocations of `transfer(recipient, amount)` that return `true` must subtract the value in `amount` from the balance of `msg.sender` and add the same value to the balance of the `recipient` address.

Specification:

```
[](willSucceed(contract.transfer(to, value), to != msg.sender
   && _balances[to] >= 0 && value >= 0
   && _balances[to] + value <= type(uint256).max
   && _balances[msg.sender] >= 0 && _balances[msg.sender] <= type(uint256).max)
      ==> <>(finished(contract.transfer(to, value), return
             ==> _balances[msg.sender] == old(_balances[msg.sender]) - value
                 && _balances[to] == old(_balances[to]) + value)))
```

erc20-transfer-correct-amount-self

Function `transfer` Transfers the Correct Amount in Self Transfers.

All non-reverting invocations of `transfer(recipient, amount)` that return `true` and where the `recipient` address equals `msg.sender` (i.e. self-transfers) must not change the balance of address `msg.sender` .

Specification:

```
[](willSucceed(contract.transfer(to, value), to == msg.sender
  && _balances[to] >= 0 && _balances[to] <= type(uint256).max)
     ==> <>(finished(contract.transfer(to, value), return
         ==> _balances[to] == old(_balances[to]))))
```

erc20-transfer-change-state

Function `transfer` Has No Unexpected State Changes.

All non-reverting invocations of `transfer(recipient, amount)` that return `true` must only modify the balance entries of the `msg.sender` and the `recipient` addresses.

Specification:

```
[](willSucceed(contract.transfer(to, value), p1 != msg.sender && p1 != to)
  ==> <>(finished(contract.transfer(to, value), return
      ==> (_totalSupply == old(_totalSupply) && _allowances == old(_allowances)
           && _balances[p1] == old(_balances[p1])  ))))
```

erc20-transfer-exceed-balance

Function `transfer` Fails if Requested Amount Exceeds Available Balance.

Any transfer of an amount of tokens that exceeds the balance of `msg.sender` must fail.

Specification:

```
[](started(contract.transfer(to, value), value > _balances[msg.sender]
   && _balances[msg.sender] >= 0 && value <= type(uint256).max)
   ==> <>(reverted(contract.transfer) || finished(contract.transfer(to, value),
          !return)))
```

erc20-transfer-recipient-overflow

Function `transfer` Prevents Overflows in the Recipient's Balance.

Any invocation of `transfer(recipient, amount)` must fail if it causes the balance of the `recipient` address to overflow.

Specification:

```
[](started(contract.transfer(to, value), to != msg.sender
   && _balances[to] + value > type(uint256).max
   && _balances[to] >= 0 && _balances[to] <= type(uint256).max
   && _balances[msg.sender] <= type(uint256).max
   && value > 0 && value <= _balances[msg.sender])
   ==> <>(reverted(contract.transfer) || finished(contract.transfer(to, value),
          !return) || finished(contract.transfer(to, value), _balances[to]
                     > old(_balances[to]) + value - type(uint256).max - 1)))
```

erc20-transfer-false

If Function `transfer` Returns `false` , the Contract State Has Not Been Changed.

If the `transfer` function in contract `contract` fails by returning `false` , it must undo all state changes it incurred before returning to the caller.

Specification:

```
[](willSucceed(contract.transfer(to, value))
   ==> <>(finished(contract.transfer(to, value), !return]
   ==> (_balances == old(_balances) && _totalSupply == old(_totalSupply)
                                    && _allowances == old(_allowances)  ))))
```

erc20-transfer-never-return-false

Function `transfe` Never Returns `false`.

The transfer function must never return `false` to signal a failure.

Specification:

```
[](!(finished(contract.transfer, !return)))
```

Properties for ERC-20 function `transferFrom`

erc20-transferfrom-revert-from-zero

Function `transferFrom` Fails for Transfers From the Zero Address.

All calls of the form `transferFrom(from, dest, amount)` where the `from` address is zero, must fail.

Specification:

```
[](started(contract.transferFrom(from, to, value), from == address(0))
   ==> <>(reverted(contract.transferFrom) || finished(contract.transferFrom,
          !return)))
```

erc20-transferfrom-revert-to-zero

Function `transferFrom` Fails for Transfers To the Zero Address.

All calls of the form `transferFrom(from, dest, amount)` where the `dest` address is zero, must fail.

Specification:

```
[](started(contract.transferFrom(from, to, value), to == address(0))
   ==> <>(reverted(contract.transferFrom) || finished(contract.transferFrom,
          !return)))
```

erc20-transferfrom-succeed-normal

Function `transferFrom` Succeeds on Admissible Non-self Transfers. All invocations of `transferFrom(from, dest, amount)` must succeed and return `true` if

- the value of `amount` does not exceed the balance of address `from`,

- the value of `amount` does not exceed the allowance of `msg.sender` for address `from`,
- transferring a value of `amount` to the address in `dest` does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call.

Specification:

```
[](started(contract.transferFrom(from, to, value), from != address(0)
   && to != address(0) && from != to && value <= _balances[from]
   && value <= _allowances[from][msg.sender]
   && _balances[to] + value <= type(uint256).max
   && value >= 0 && _balances[to] >= 0 && _balances[from] >= 0
   && _balances[from] <= type(uint256).max
   && _allowances[from][msg.sender] >= 0
   && _allowances[from][msg.sender] <= type(uint256).max)
      ==> <>(finished(contract.transferFrom(from, to, value), return)))
```

erc20-transferfrom-succeed-self

Function `transferFrom` Succeeds on Admissible Self Transfers.

All invocations of `transferFrom(from, dest, amount)` where the `dest` address equals the `from` address (i.e. self-transfers) must succeed and return `true` if:

- The value of `amount` does not exceed the balance of address `from`,
- the value of `amount` does not exceed the allowance of `msg.sender` for address `from`, and
- the supplied gas suffices to complete the call.

Specification:

```
[](started(contract.transferFrom(from, to, value), from != address(0)
   && from == to && value <= _balances[from]
   && value <= _allowances[from][msg.sender]
   && value >= 0 && _balances[from] <= type(uint256).max
   && _allowances[from][msg.sender] <= type(uint256).max)
     ==> <>(finished(contract.transferFrom(from, to, value), return)))
```

erc20-transferfrom-correct-amount

Function `transferFrom` Transfers the Correct Amount in Non-self Transfers.

All invocations of `transferFrom(from, dest, amount)` that succeed and that return `true` subtract the value in `amount` from the balance of address `from` and add the same value to the balance of address `dest`.

Specification:

```
[](willSucceed(contract.transferFrom(from, to, value), from != to && value >= 0
&& _balances[from] >= 0 && _balances[from] <= type(uint256).max
&& _balances[to] >= 0 && _balances[to] + value <= type(uint256).max)
   ==> <>(finished(contract.transferFrom(from, to, value), return
          ==> _balances[from] == old(_balances[from]) - value
              && _balances[to] == old(_balances[to] + value))))
```

erc20-transferfrom-correct-amount-self

Function `transferFrom` Performs Self Transfers Correctly.

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` and where the address in `from` equals the address in `dest` (i.e. self-transfers) do not change the balance entry of the `from` address (which equals `dest`).

Specification:

```
[](willSucceed(contract.transferFrom(from, to, value), from == to
   && value >= 0 && value <= type(uint256).max && _balances[from] >= 0
   && _balances[from] <= type(uint256).max)
      ==> <>(finished(contract.transferFrom(from, to, value), return
             ==> _balances[from] == old(_balances[from]))))
```

erc20-transferfrom-correct-allowance

Function `transferFrom` Updated the Allowance Correctly.

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` must decrease the allowance for address `msg.sender` over address `from` by the value in `amount`.

Specification:

```
[](willSucceed(contract.transferFrom(from, to, value), value >= 0
   && value <= type(uint256).max && _balances[from] >= 0
   && _balances[from] <= type(uint256).max && _balances[to] >= 0
   && _balances[to] <= type(uint256).max && _allowances[from][msg.sender] >= 0
   && _allowances[from][msg.sender] <= type(uint256).max)
      ==> <>(finished(contract.transferFrom(from, to, value), return
          ==> ((_allowances[from][msg.sender]
                == old(_allowances[from][msg.sender]) - value)
               || (_allowances[from][msg.sender]
                   == old(_allowances[from][msg.sender])
                      && (from == msg.sender
                          || old(_allowances[from][msg.sender])
                             == type(uint256).max))))))
```

erc20-transferfrom-change-state

Function `transferFrom` Has No Unexpected State Changes.

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` may only modify the following state variables:

- The balance entry for the address in `dest`,
- The balance entry for the address in `from`,
- The allowance for the address in `msg.sender` for the address in `from`. Specification:

```
[](willSucceed(contract.transferFrom(from, to, amount), p1 != from && p1 != to
   && (p2 != from || p3 != msg.sender))
    ==> <>(finished(contract.transferFrom(from, to, amount), return
     ==> (_totalSupply == old(_totalSupply) && _balances[p1] == old(_balances[p1])
         && _allowances[p2][p3] == old(_allowances[p2][p3])  ))))
```

erc20-transferfrom-fail-exceed-balance

Function `transferFrom` Fails if the Requested Amount Exceeds the Available Balance.

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the balance of address `from` must fail.

Specification:

```
[](started(contract.transferFrom(from, to, value), value > _balances[from]
   && _balances[from] >= 0 && _balances[from] <= type(uint256).max)
       ==> <>(reverted(contract.transferFrom)
              || finished(contract.transferFrom, !return)))
```

erc20-transferfrom-fail-exceed-allowance

Function `transferFrom` Fails if the Requested Amount Exceeds the Available Allowance.

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the allowance of address `msg.sender` must fail.

Specification:

```
  [](started(contract.transferFrom(from, to, value), value > _allowances[from]
[msg.sender]
    && _allowances[from][msg.sender] >= 0 && value <= type(uint256).max)
    ==> <>(reverted(contract.transferFrom)
          || finished(contract.transferFrom(from, to, value), !return)
          || finished(contract.transferFrom(from, to, value), return
             && (msg.sender == from
                 || _allowances[from][msg.sender] == type(uint256).max))))
```

erc20-transferfrom-fail-recipient-overflow

Function `transferFrom` Prevents Overflows in the Recipient's Balance.

Any call of `transferFrom(from, dest, amount)` with a value in `amount` whose transfer would cause an overflow of the balance of address `dest` must fail.

Specification:

```
[](started(contract.transferFrom(from, to, value), from != to
   && _balances[to] + value > type(uint256).max && value <= type(uint256).max
   && _balances[to] >= 0 && _balances[to] <= type(uint256).max)
      ==> <>(reverted(contract.transferFrom)
          || finished(contract.transferFrom(from, to, value), !return)
          || finished(contract.transferFrom(from, to, value), _balances[to]
              > old(_balances[to]) + value - type(uint256).max - 1)))
```

erc20-transferfrom-false

If Function `transferFrom` Returns `false`, the Contract's State Has Not Been Changed.

If `transferFrom` returns `false` to signal a failure, it must undo all incurred state changes before returning to the caller.

Specification:

```
[](willSucceed(contract.transfer(to, value))
   ==> <>(finished(contract.transfer(to, value), !return
   ==> (_balances == old(_balances) && _totalSupply == old(_totalSupply)
                                    && _allowances == old(_allowances)  ))))
```

erc20-transferfrom-never-return-false

Function `transferFrom` Never Returns `false`.

The `transferFrom` function must never return `false`.

Specification:

```
[](!(finished(contract.transferFrom, !return)))
```

Properties related to function `totalSupply`

erc20-totalsupply-succeed-always

Function `totalSupply` Always Succeeds.

The function `totalSupply` must always succeeds, assuming that its execution does not run out of gas.

Specification:

```
[](started(contract.totalSupply) ==> <>(finished(contract.totalSupply)))
```

erc20-totalsupply-correct-value

Function `totalSupply` Returns the Value of the Corresponding State Variable.

The `totalSupply` function must return the value that is held in the corresponding state variable of contract contract.

Specification:

```
[](willSucceed(contract.totalSupply)
   ==> <>(finished(contract.totalSupply, return == _totalSupply)))
```

erc20-totalsupply-change-state

Function `totalSupply` Does Not Change the Contract's State.

The `totalSupply` function in contract contract must not change any state variables.

Specification:

```
[](willSucceed(contract.totalSupply)
   ==> <>(finished(contract.totalSupply, _totalSupply == old(_totalSupply)
          && _balances == old(_balances) && _allowances == old(_allowances)  )))
```

Properties related to function `balanceOf`

erc20-balanceof-succeed-always

Function `balanceOf` Always Succeeds.

Function `balanceOf` must always succeed if it does not run out of gas.

Specification:

```
[](started(contract.balanceOf) ==> <>(finished(contract.balanceOf)))
```

erc20-balanceof-correct-value

Function `balanceOf` Returns the Correct Value.

Invocations of `balanceOf(owner)` must return the value that is held in the contract's balance mapping for address `owner`.

Specification:

```
[](willSucceed(contract.balanceOf)
   ==> <>(finished(contract.balanceOf(owner), return == _balances[owner])))
```

erc20-balanceof-change-state

Function `balanceOf` Does Not Change the Contract's State.

Function `balanceOf` must not change any of the contract's state variables.

Specification:

```
[](willSucceed(contract.balanceOf)
   ==> <>(finished(contract.balanceOf(owner), _totalSupply == old(_totalSupply)
          && _balances == old(_balances)
          && _allowances == old(_allowances)  )))
```

Properties related to function `allowance`

erc20-allowance-succeed-always

Function `allowance` Always Succeeds.

Function `allowance` must always succeed, assuming that its execution does not run out of gas.

Specification:

```
[](started(contract.allowance) ==> <>(finished(contract.allowance)))
```

erc20-allowance-correct-value

Function `allowance` Returns Correct Value.

Invocations of `allowance(owner, spender)` must return the allowance that address `spender` has over tokens held by address `owner`.

Specification:

```
[](willSucceed(contract.allowance(owner, spender))
   ==> <>(finished(contract.allowance(owner, spender),
           return == _allowances[owner][spender])))
```

erc20-allowance-change-state

Function `allowance` Does Not Change the Contract's State.

Function `allowance` must not change any of the contract's state variables.

Specification:

```
[](willSucceed(contract.allowance(owner, spender))
   ==> <>(finished(contract.allowance(owner, spender),
   _totalSupply == old(_totalSupply) && _balances == old(_balances)
               && _allowances == old(_allowances)  )))
```

Properties related to function `approve`

erc20-approve-revert-zero

Function `approve` Prevents Giving Approvals For the Zero Address.

All calls of the form `approve(spender, amount)` must fail if the address in `spender` is the zero address.

Specification:

```
[](started(contract.approve(spender, value), spender == address(0))
   ==> <>(reverted(contract.approve)
          || finished(contract.approve(spender, value), !return)))
```

erc20-approve-succeed-normal

Function `approve` Succeeds for Admissible Inputs.

All calls of the form `approve(spender, amount)` must succeed, if

- the address in `spender` is not the zero address and
- the execution does not run out of gas.

Specification:

```
[](started(contract.approve(spender, value), spender != address(0))
   ==> <>(finished(contract.approve(spender, value), return)))
```

erc20-approve-correct-amount

Function `approve` Updates the Approval Mapping Correctly.

All non-reverting calls of the form `approve(spender, amount)` that return `true` must correctly update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount`.

Specification:

```
[](willSucceed(contract.approve(spender, value), spender != address(0)
   && value >= 0 && value <= type(uint256).max)
       ==> <>(finished(contract.approve(spender, value), return
              ==> _allowances[msg.sender][spender] == value)))
```

erc20-approve-change-state

Function `approve` Has No Unexpected State Changes.

All calls of the form `approve(spender, amount)` must only update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount` and incur no other state changes.

Specification:

```
[](willSucceed(contract.approve(spender, value), spender != address(0)
   && (p1 != msg.sender || p2 != spender))
       ==> <>(finished(contract.approve(spender, value), return
              ==> _totalSupply == old(_totalSupply) && _balances == old(_balances)
                  && _allowances[p1][p2] == old(_allowances[p1][p2])  )))
```

erc20-approve-false

If Function `approve` Returns `false`, the Contract's State Has Not Been Changed.

If function `approve` returns `false` to signal a failure, it must undo all state changes that it incurred before returning to the caller.

Specification:

```
[](willSucceed(contract.approve(spender, value))
   ==> <>(finished(contract.approve(spender, value), !return
          ==> (_balances == old(_balances) && _totalSupply == old(_totalSupply)
               && _allowances == old(_allowances)  ))))
```

erc20-approve-never-return-false

Function `approve` Never Returns `false`.

The function `approve` must never returns `false`.

Specification:

```
 [](!(finished(contract.approve, !return)))
```

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.