CERTIK

Security Assessment

Rat Alert

May 10th, 2022

Table of Contents

Summary

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

- 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.

Overview

Project Summary

Project Name	Rat Alert
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/ratalert/ratalert-contracts
Commit	57ce8c6666b5184d60c710227a2e5784aaa83c43

Audit Summary

Delivery Date	May 10, 2022 UTC
Audit Methodology	Static Analysis, Manual Review

Vulnerability Summary

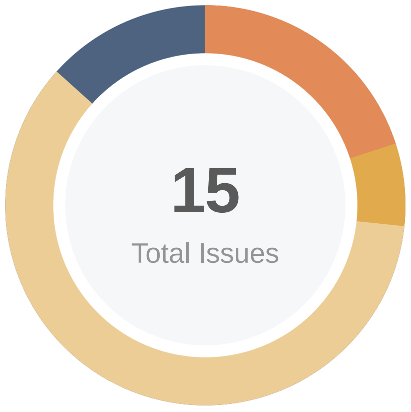
Vulnerability Level	Total	Pending	Declined	Acknowledged	Mitigated	Partially Resolved	Resolved
● Critical	0	0	0	0	0	0	0
● Major	3	0	0	0	2	0	1
● Medium	1	0	0	0	0	0	1
● Minor	9	0	0	1	0	2	6
● Informational	2	0	0	0	0	1	1
● Discussion	0	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
CON	contracts/Controllable.sol	69ad10ad0ecd184b39899edee1a6e182a6ae3b883d97bb0cd5d4a24b672503a2
TRA	contracts/Traits.sol	68656679305bb1cfee19516ed8d143d353b2af2e1468833d078a90991140c888
ICU	contracts/IClaim.sol	d5aebff440e6ad8b430614ec929d2dbb72dadb13789af6feb12f0c7db3cc3dfb
CUB	contracts/ControllableUpgradeable.sol	efa80fc88b9a0ddbb6214d49a5efdbe2b790bccd96f079f72244a20852a5c58c
LSB	contracts/LeStake.sol	8261e498ab8985b238462cfe7972fea472d56efec9df19c435da75b31a2cb03d
VRF	contracts/VRFConsumer.sol	132389ee746d945b1e435d8a735136b62e53bb54a310fe20766f296d41686873
IVB	contracts/IVenue.sol	7c1d776058e39dda15be8625f308499166e7d9211d65528f1bb33ec30eea138d
IPB	contracts/IPaywall.sol	66b5ebbc50573506a1ac360193196f30fb21061de14a383699d956ce1129068e
EKB	contracts/EntrepreneurKitchen.sol	c4a9c605f291ee8c49a4e89f172af3ae196bb1c85d890105be2d1efa93a3d9e1
ICB	contracts/ICharacter.sol	f62d78423919613652cd24fa4f0777bc350a2a8c50c1904482d2c3e671fed4ea
CLA	contracts/Claim.sol	aa41c52a156bcfa57c1443e7c37ee5b976b9a8e462d76a39de7b2251f3f41a5c
KSB	contracts/KitchenShop.sol	011008db9dcd03ed5f67747b19d97ca585936734afaf9ab1352808b3ce15052d
MIN	contracts/Mint.sol	4d019daf5916beecf07c31a2d1ca00ad9f27c0db39da7f716cf8b1b727e98bbf
PRO	contracts/Properties.sol	47949b52c403ef5ca52a1702a6802a08198b3b83ce4131e103e949bf3c3c3111
GFB	contracts/GourmetFood.sol	e1ad10588b20c31a49fac3d3048cde2ce97410f509a7b2df48b5f2a94eb577f6
CFB	contracts/CasualFood.sol	ecbda784e8ed8cb347361ddf93b1b608aad7c9139c3a5a4b5a093ceb696cb01c
VEN	contracts/Venue.sol	7f5151c2042eb9e9455102aa5a66126838c4ad37adddff146355a822cc21eef8
CHA	contracts/Character.sol	cc08ae03915e3c139da1ea59cc6a5e5ed738bb2854dbacfcc33de0815815389c
GPB	contracts/GenericPausable.sol	02ef95404a88e92c33235f709bc747699a5f3e2215c694b8ec1b8427a5f864a1
ITB	contracts/ITraits.sol	a428f54a7c6a5cc439913bcf71fc541bf30174a4e1f478db7aa05691ed0d579e
PAY	contracts/Paywall.sol	f0ed079e3915bc4648c619644daedf1b1dc601fa7a67665201d5c6cf594a5dcc
IPU	contracts/IProperties.sol	585b84364910d51beae2866a922b54f4dcb5bd83c8fba788be527e521af545e9

ID	File	SHA256 Checksum
VRC	contracts/VRFCoordinatorMock.sol	be7c9fb4752e8f264cb6ffd99cf1caf1904843e8a877b376b3af8118a1014741
FOO	contracts/Food.sol	e9dd3592d262a42b8d9bf9b4abb3f33f90143fc1486ae7f0be896244764a8469
LTM	contracts/LinkTokenMock.sol	ea00954295148334cf8ff22c8930b25f9b509a1fdbe9ed095f34987eb0138ca7
MSB	contracts/McStake.sol	521bc531aece408cd0528d8f5f8b1a9eff3c6640645f4a9e5e46d3ed33d1de4c
GYM	contracts/Gym.sol	71f06ffdcf9d9c60f8ad6cca6d7a1f3a03adac443f5a654f4a70d30cabcf377d
TSB	contracts/TheStakehouse.sol	e5db840a0dd2eb23a61c40cb86d09b0285bae08c5b56b0a0d48d2ab563f1122f
KIT	contracts/Kitchen.sol	e56a4437de938e34d6a9a9d72cc13bc072da27019ed51ba13575e706716917be
FFB	contracts/FastFood.sol	9fe5b87b9774acada7dc3a0f804e8c74f28defe6aea53a41affd49dd81353711
IMB	contracts/IMint.sol	099488e9b84b1ba9fb682eb5fb7370e349a085ecc0adde68cdcb7daedbad33c1

Findings



🟥 **Critical**	**0** (0.00%)
🟧 **Major**	**3** (20.00%)
🟨 **Medium**	**1** (6.67%)
🟨 **Minor**	**9** (60.00%)
🟦 **Informational**	**2** (13.33%)
🟩 **Discussion**	**0** (0.00%)

ID	Title	Category	Severity	Status
GLOBAL-01	Centralization Risk In Proxy-admin Configuration	**Logical Issue, Centralization / Privilege**	● **Major**	Mitigated
GLOBAL-02	Centralization Related Risks	**Centralization / Privilege**	● **Major**	Mitigated
CFB-01	Inconsistency With White Paper	Logical Issue	● Minor	✓ Resolved
CHA-01	Optimizable Transfer Patterns	Volatile Code	● Major	✓ Resolved
CHA-02	Optimizable Data Structure	Volatile Code	● Minor	✓ Resolved
CLA-01	Users Do Not Pay For $link Usage	Logical Issue	● Minor	✓ Resolved
CON-01	Flawed $Kitchen Token Implementation	Logical Issue	● Medium	✓ Resolved
CON-02	Potential Reentrancy Attack	Logical Issue	● Minor	✓ Resolved
CON-03	Optimizable Emit Events Design	Coding Style	● Minor	Acknowledged
CON-04	Receive Token By `burn()`	Logical Issue	● Minor	Partially Resolved
CON-05	Bad ERC721 Token Stake Method	Volatile Code	● Minor	✓ Resolved
CON-06	Payment Token By Mint()	Logical Issue	● Minor	Partially Resolved
CON-07	Risk Of Transaction Revert Due To Gas Reaching Limit	Logical Issue	● Minor	✓ Resolved

ID	Title	Category	Severity	Status
CON-08	Improper Usage Of `public` And `external` Type	Gas Optimization	● Informational	Partially Resolved
CON-09	Using Standardized Base64 Library	Coding Style	● Informational	Resolved

GLOBAL-01 | Centralization Risk In Proxy-admin Configuration

Category	Severity	Location	Status
Logical Issue, Centralization / Privilege	● **Major**		Mitigated

Description

All of Rat Alter contracts that carry game mechanics are upgradeable, this idea behind this is to enable the RatAlert DAO to agree upon and change parameters of the game if required,and the contracts of the project are deployed with `proxy`. Apart from the logic in the specific logic contract, the contracts deployed via proxies can add additional permission controls or other logic. Since the proxy contract is not in the audit scope, it will be treated as a black box and assumed functional correctness. However, there will be potential centralization risk in the proxy:

- The admin of the proxy contract has the authority to execute any delegate call.
- Proxy-admin does not make a reasonable configuration in any code file.

Any compromise to the `admin` account may allow the hacker to take advantage of this and users' assets may suffer loss.

Recommendation

Making more sense of the Proxy-admin configuration

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:

Above all

Complete proxy-admin configuration operations in the code.

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*.

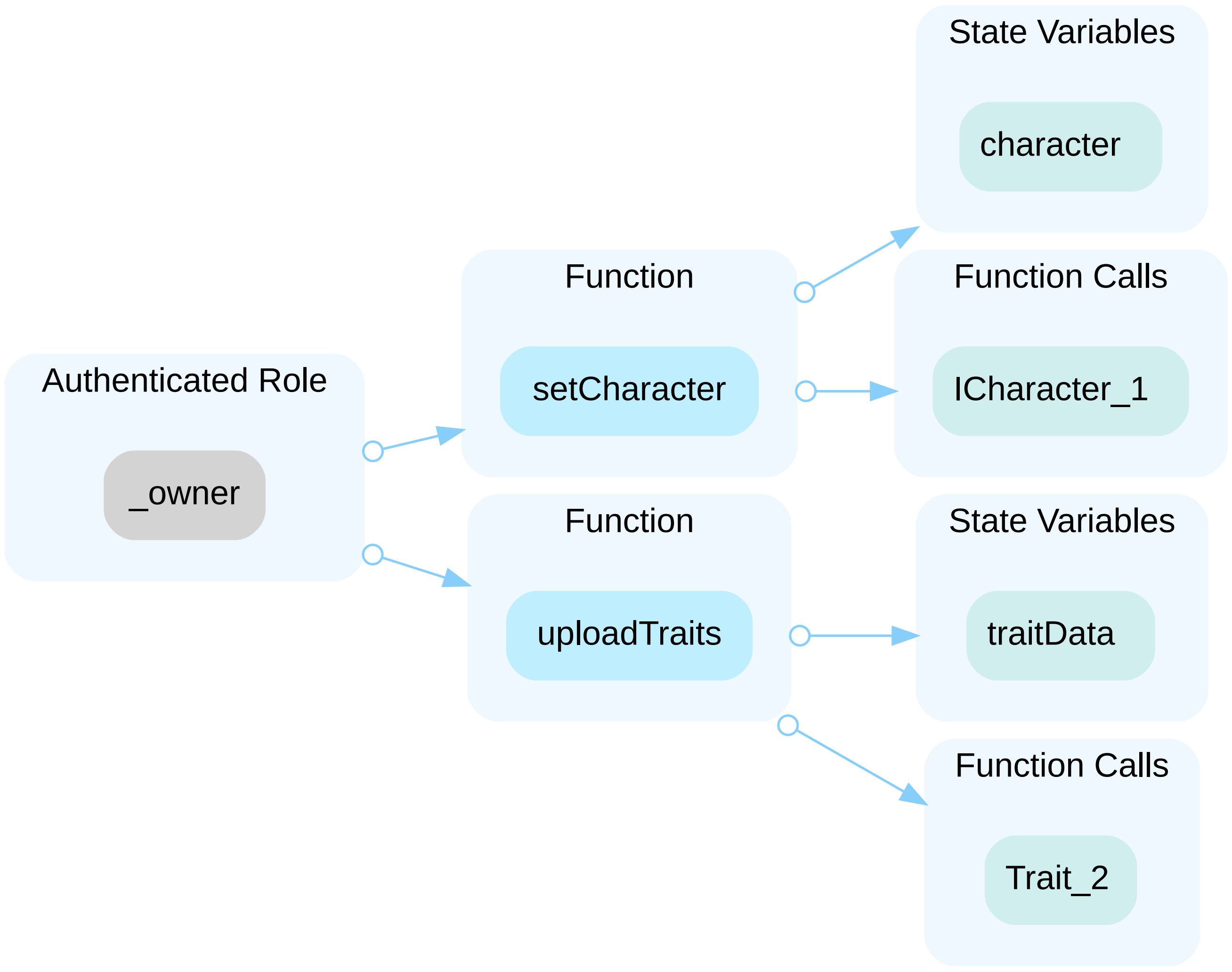
- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

GLOBAL-02 | Centralization Related Risks

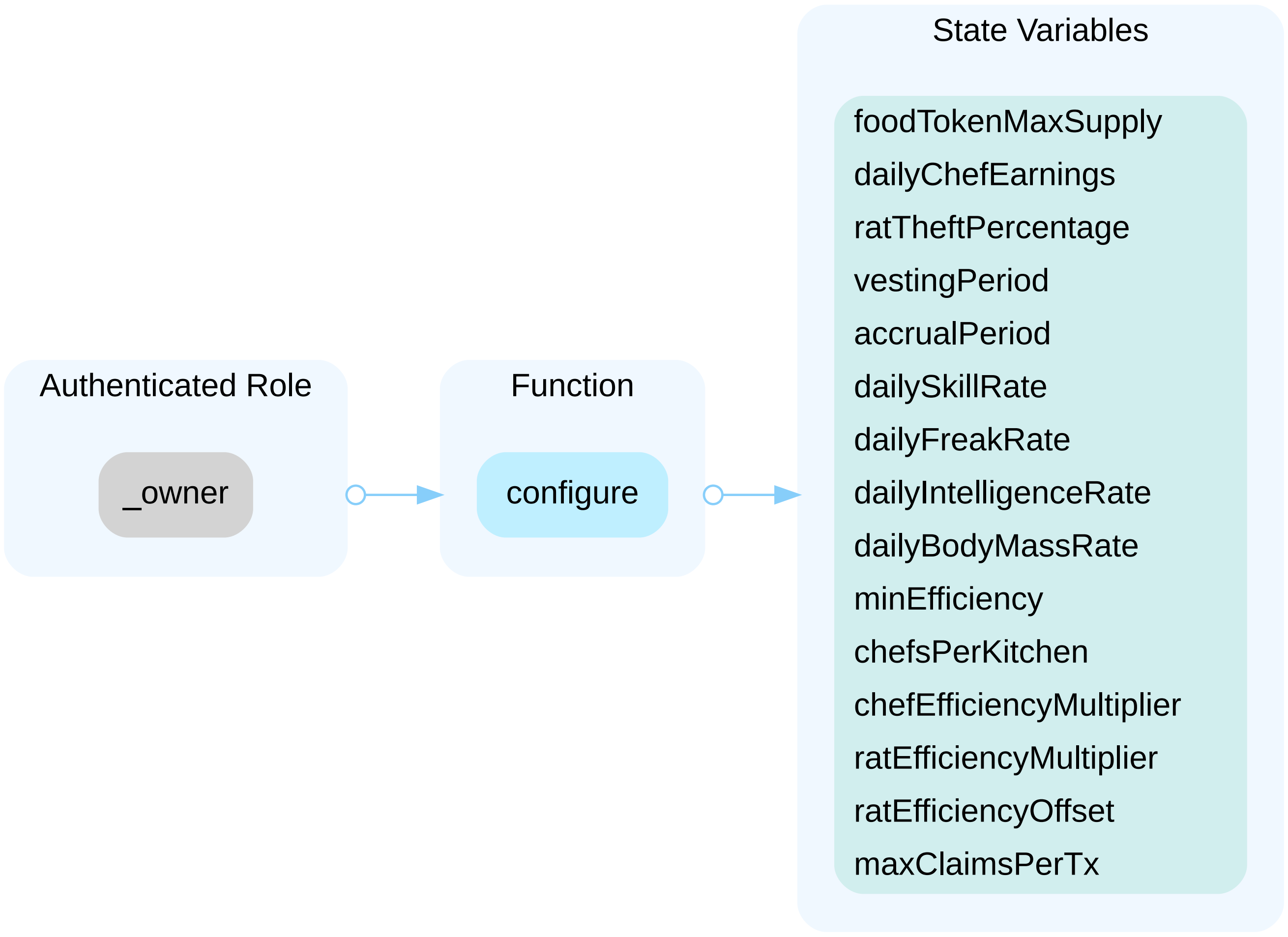
Category	Severity	Location	Status
Centralization / Privilege	● **Major**		Mitigated

Description

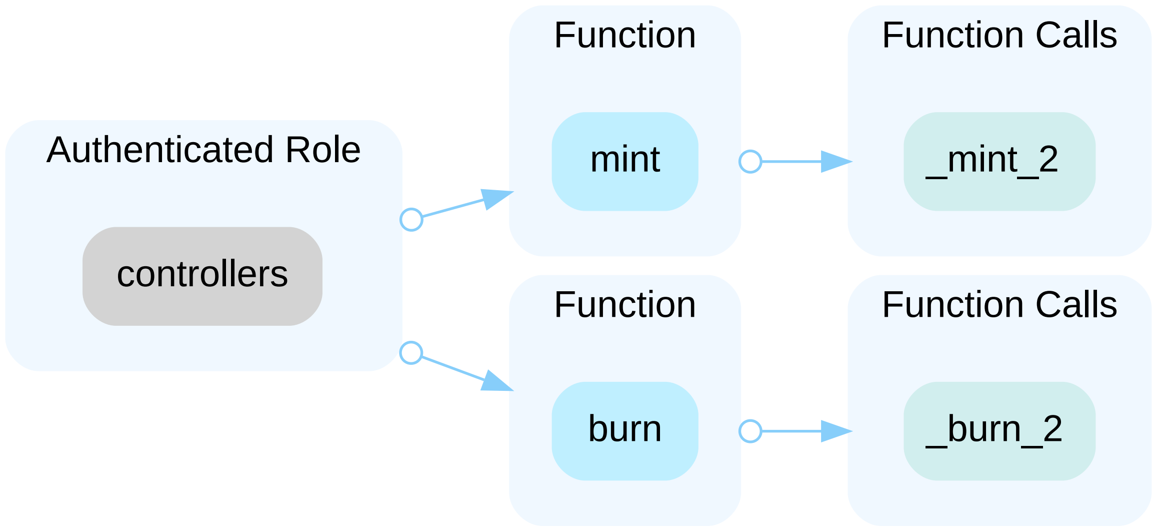
In the contract `Traits` the role `_owner` has authority over the functions shown in the diagram below.



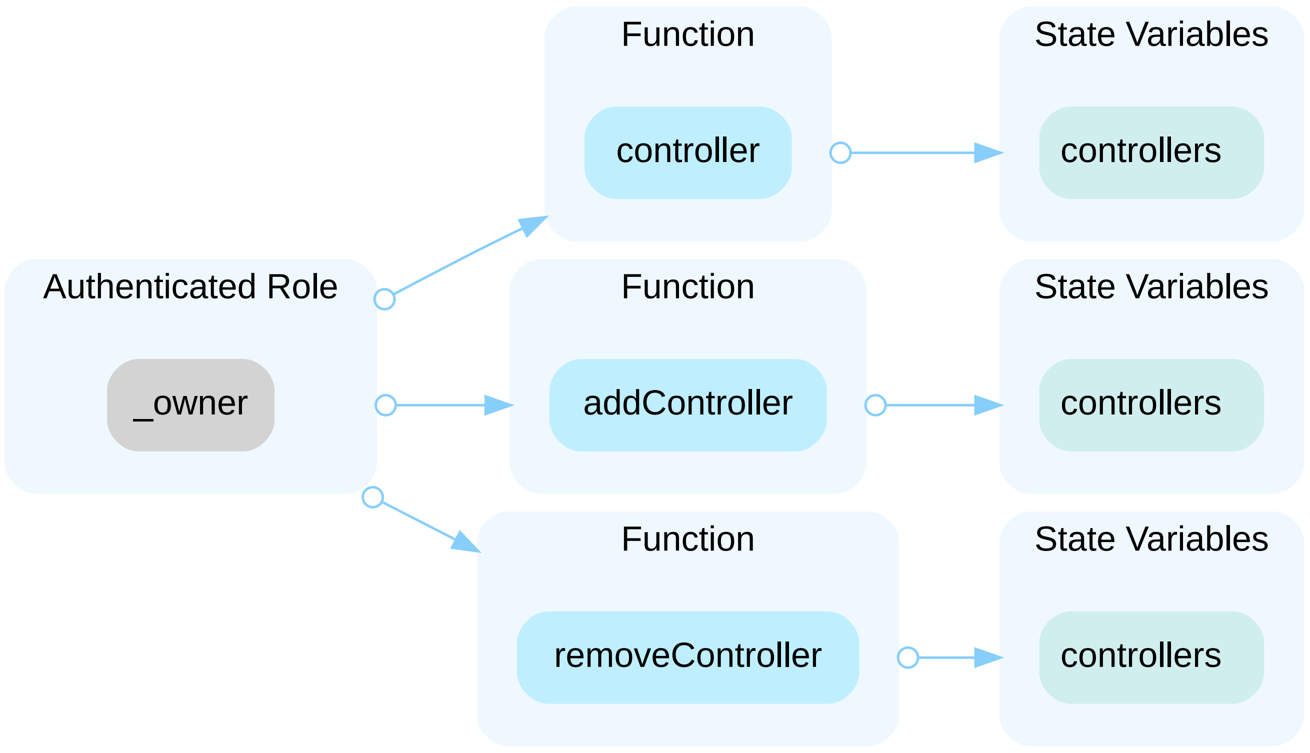
In the contract `TheStakehouse` the role `_owner` has authority over the functions shown in the diagram below.



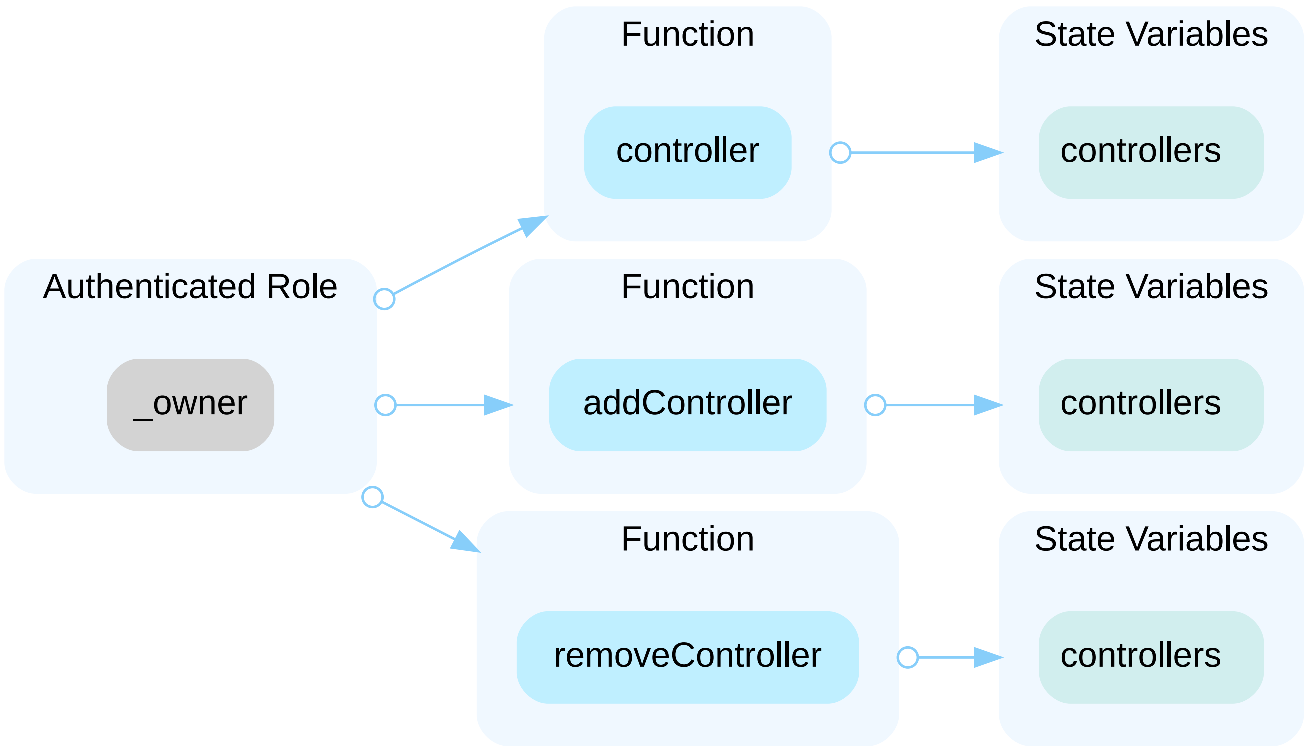
In the contract `Food` the role `controllers` has authority over the functions shown in the diagram below.



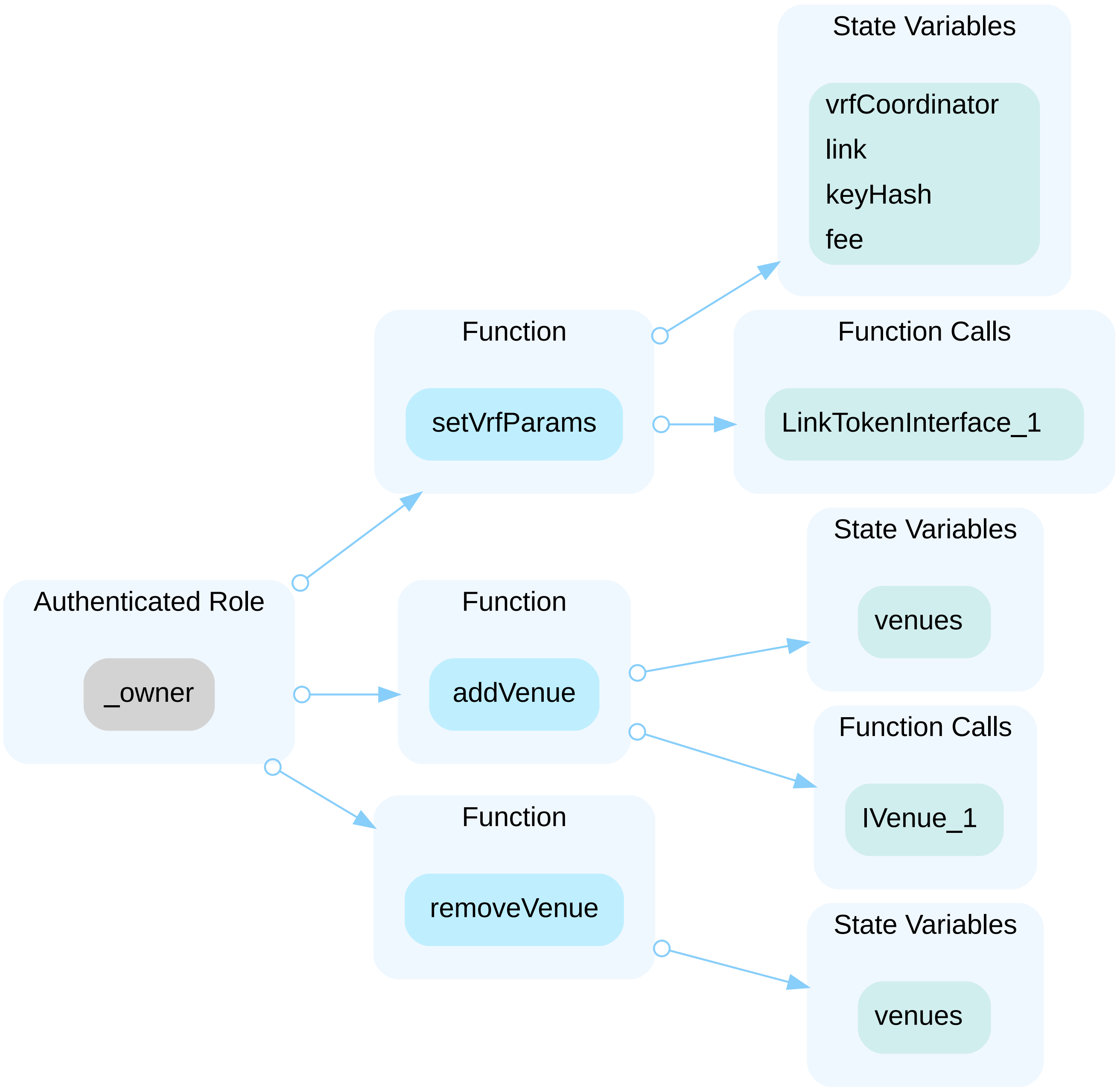
In the contract `ControllableUpgradeable` the role `_owner` has authority over the functions shown in the diagram below.



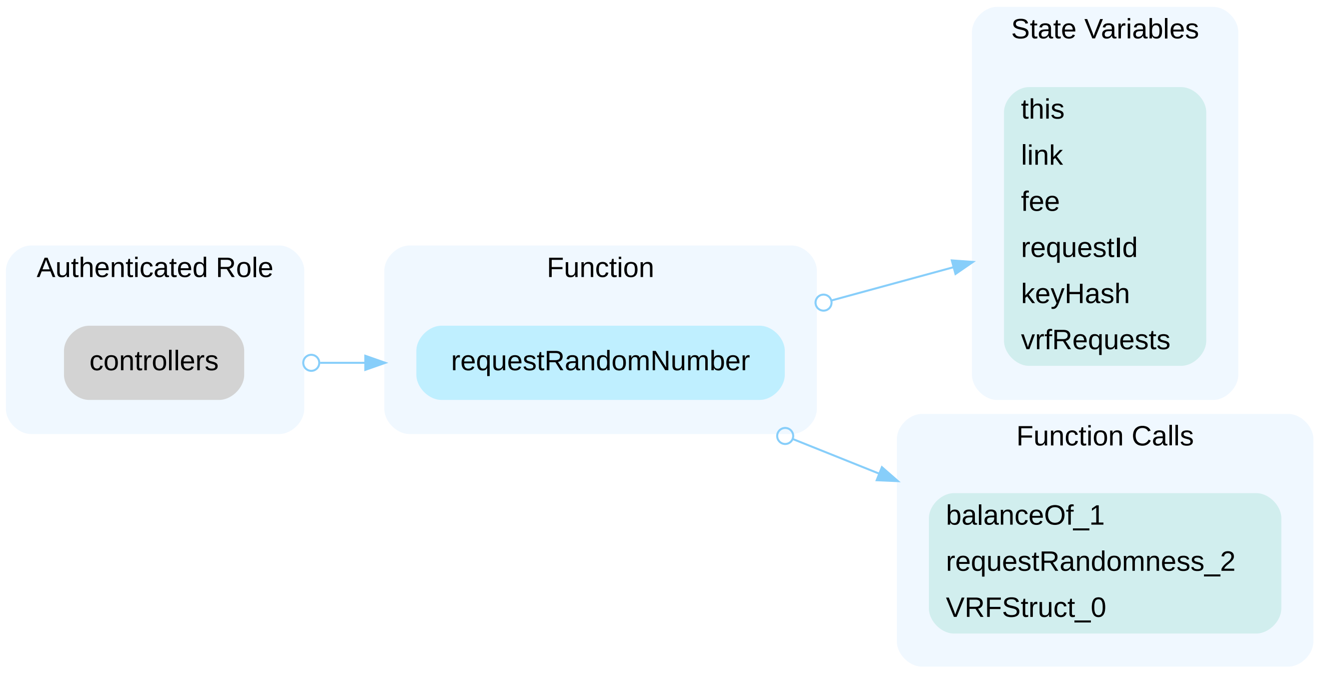
In the contract `Controllable` the role `_owner` has authority over the functions shown in the diagram below.



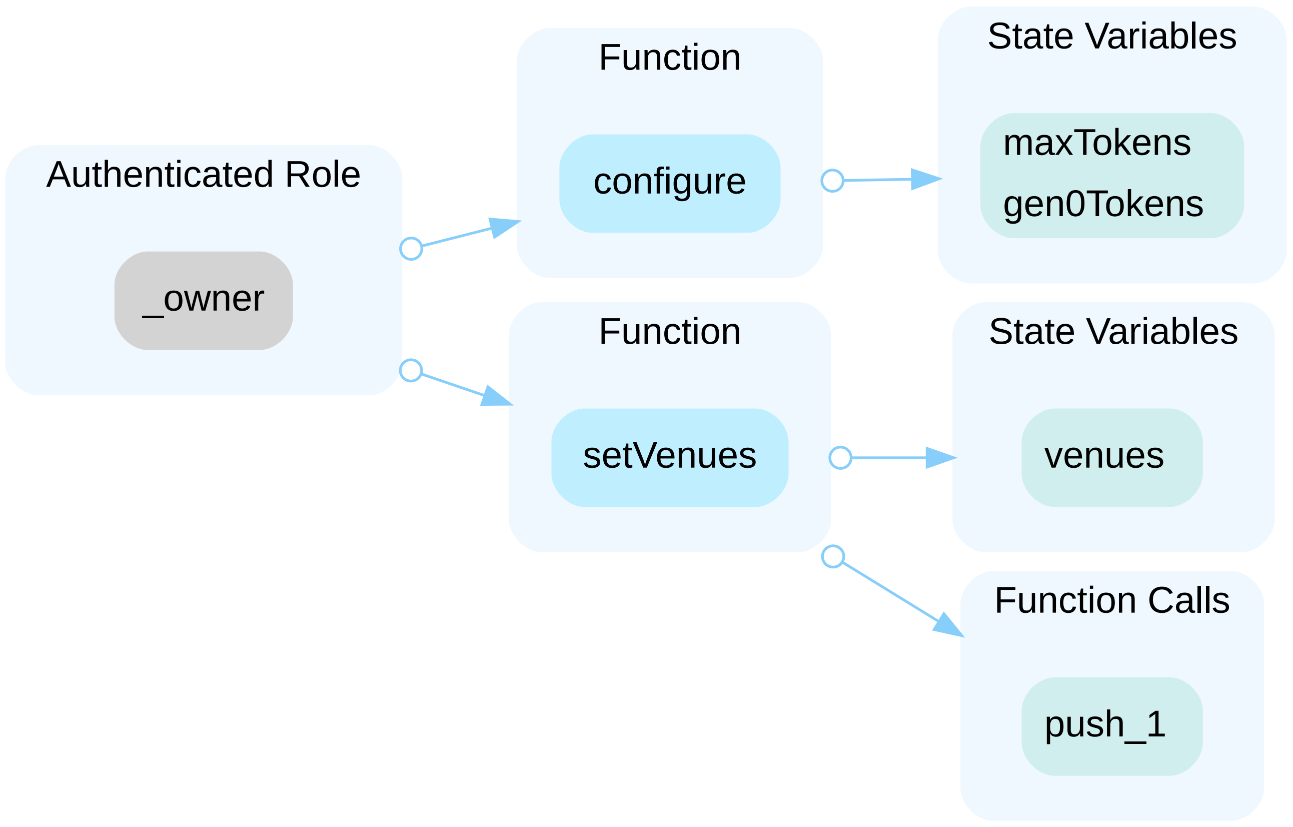
In the contract `Claim` the role `_owner` has authority over the functions shown in the diagram below.



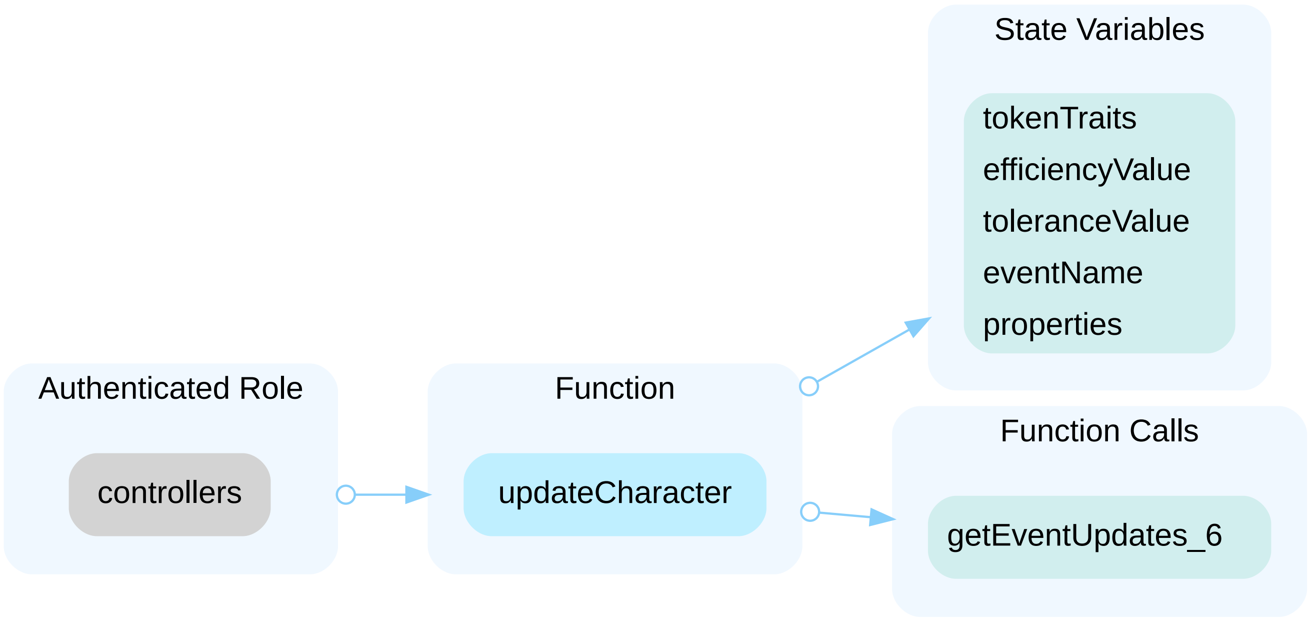
In the contract `Claim` the role `controllers` has authority over the functions shown in the diagram below.



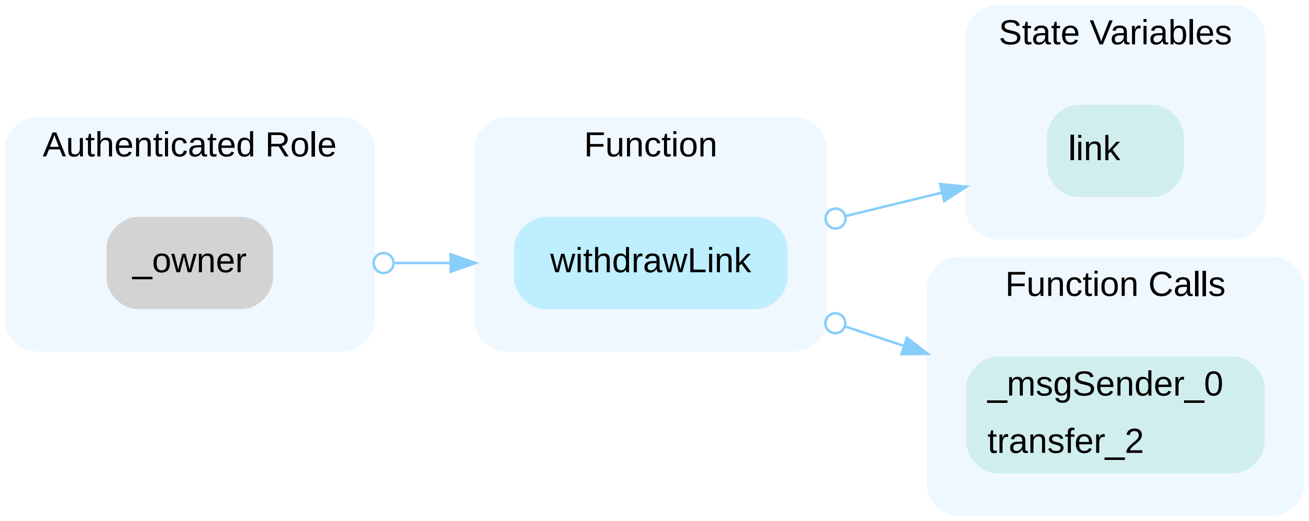
In the contract `Character` the role `_owner` has authority over the functions shown in the diagram below.



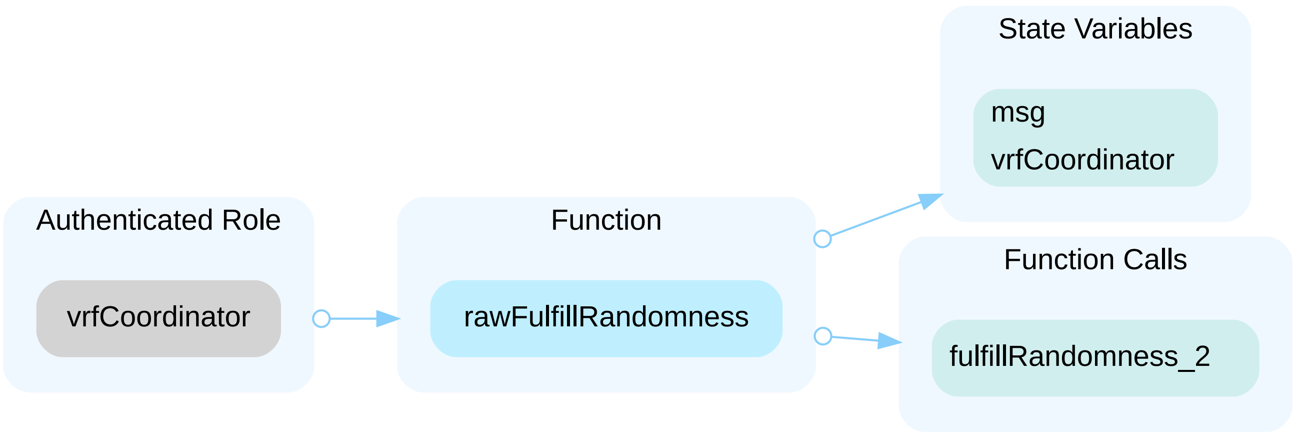
In the contract `Character` the role `controllers` has authority over the functions shown in the diagram below.



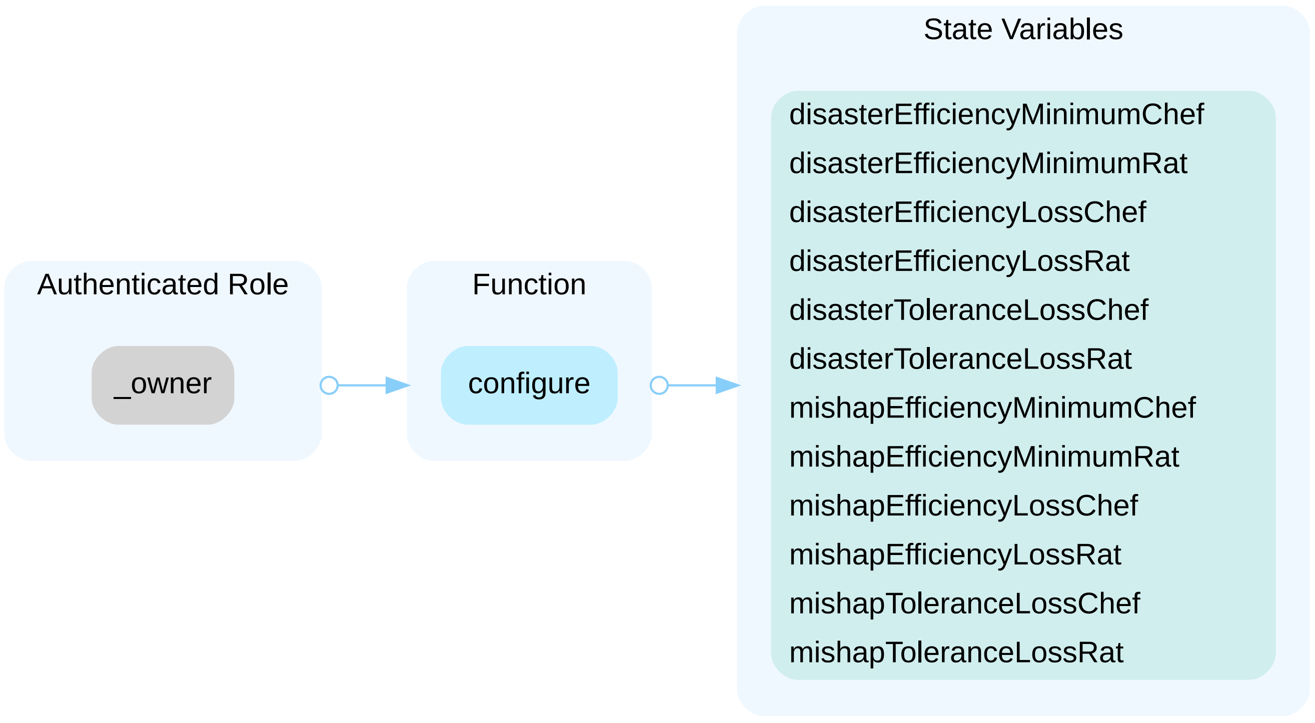
In the contract `VRFConsumer` the role `_owner` has authority over the functions shown in the diagram below.



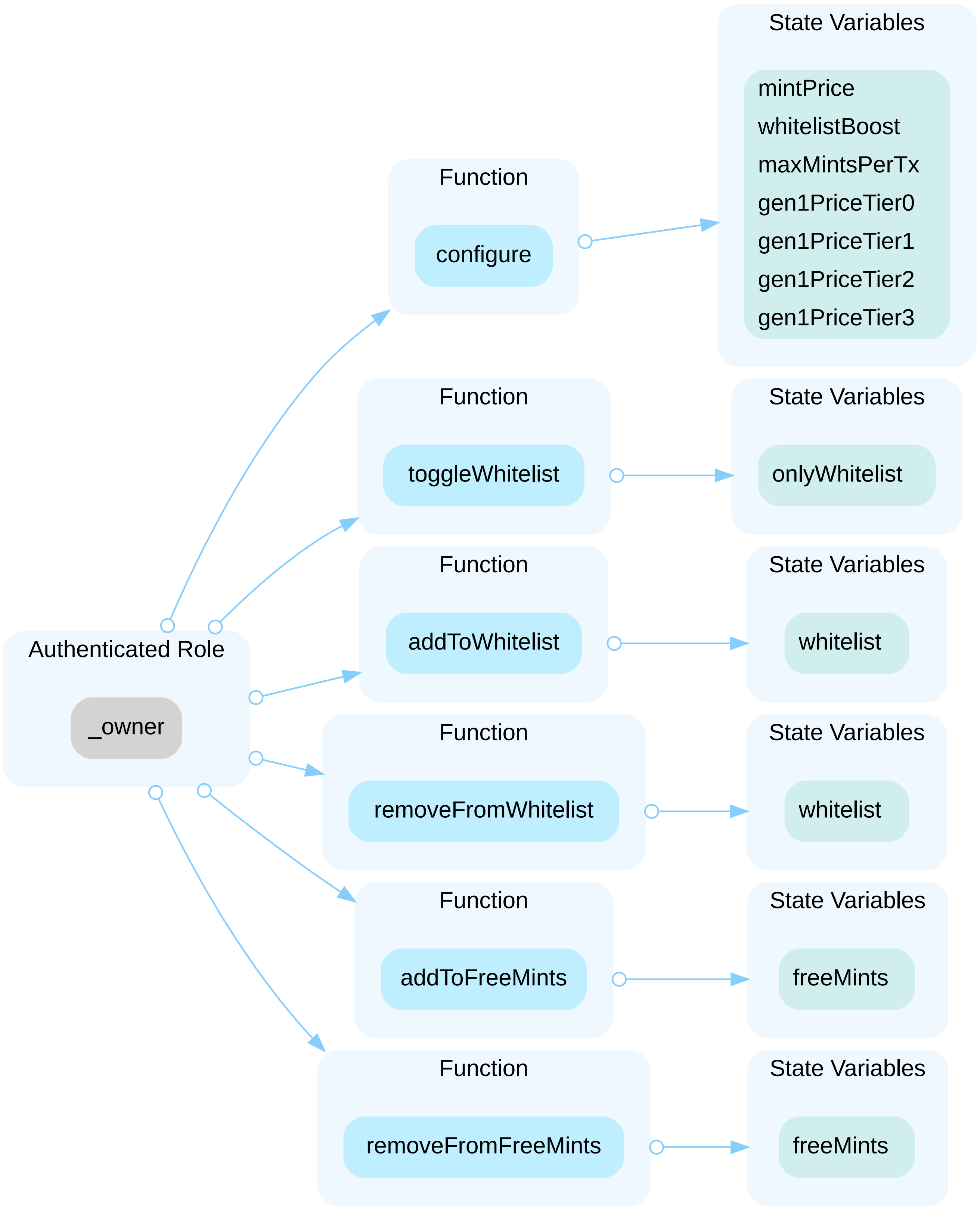
In the contract `VRFConsumer` the role `vrfCoordinator` has authority over the functions shown in the diagram below.



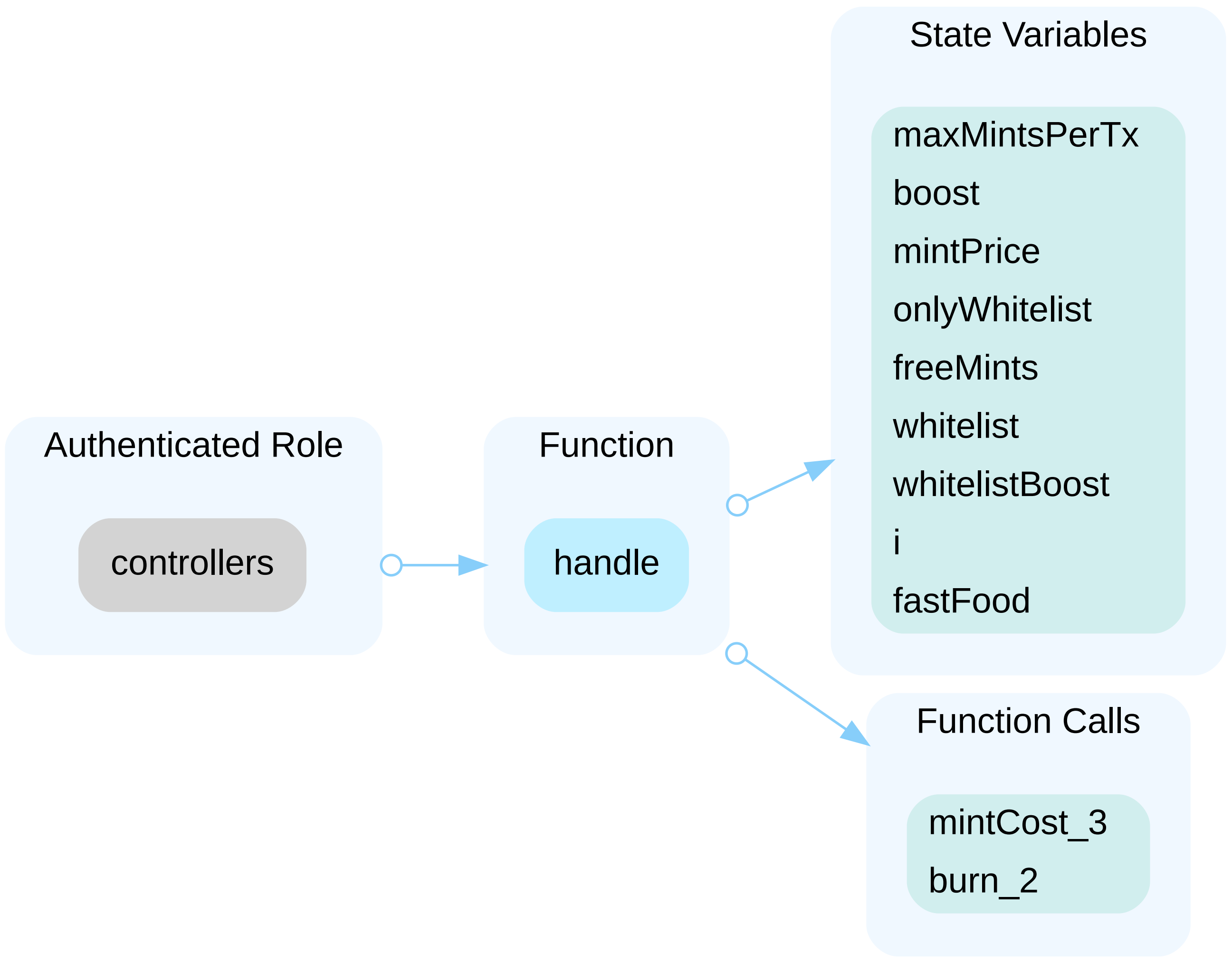
In the contract `Properties` the role `_owner` has authority over the functions shown in the diagram below.


Authenticated Role
_owner
Function
configure
State Variables
disasterEfficiencyMinimumChef
disasterEfficiencyMinimumRat
disasterEfficiencyLossChef
disasterEfficiencyLossRat
disasterToleranceLossChef
disasterToleranceLossRat
mishapEfficiencyMinimumChef
mishapEfficiencyMinimumRat
mishapEfficiencyLossChef
mishapEfficiencyLossRat
mishapToleranceLossChef
mishapToleranceLossRat

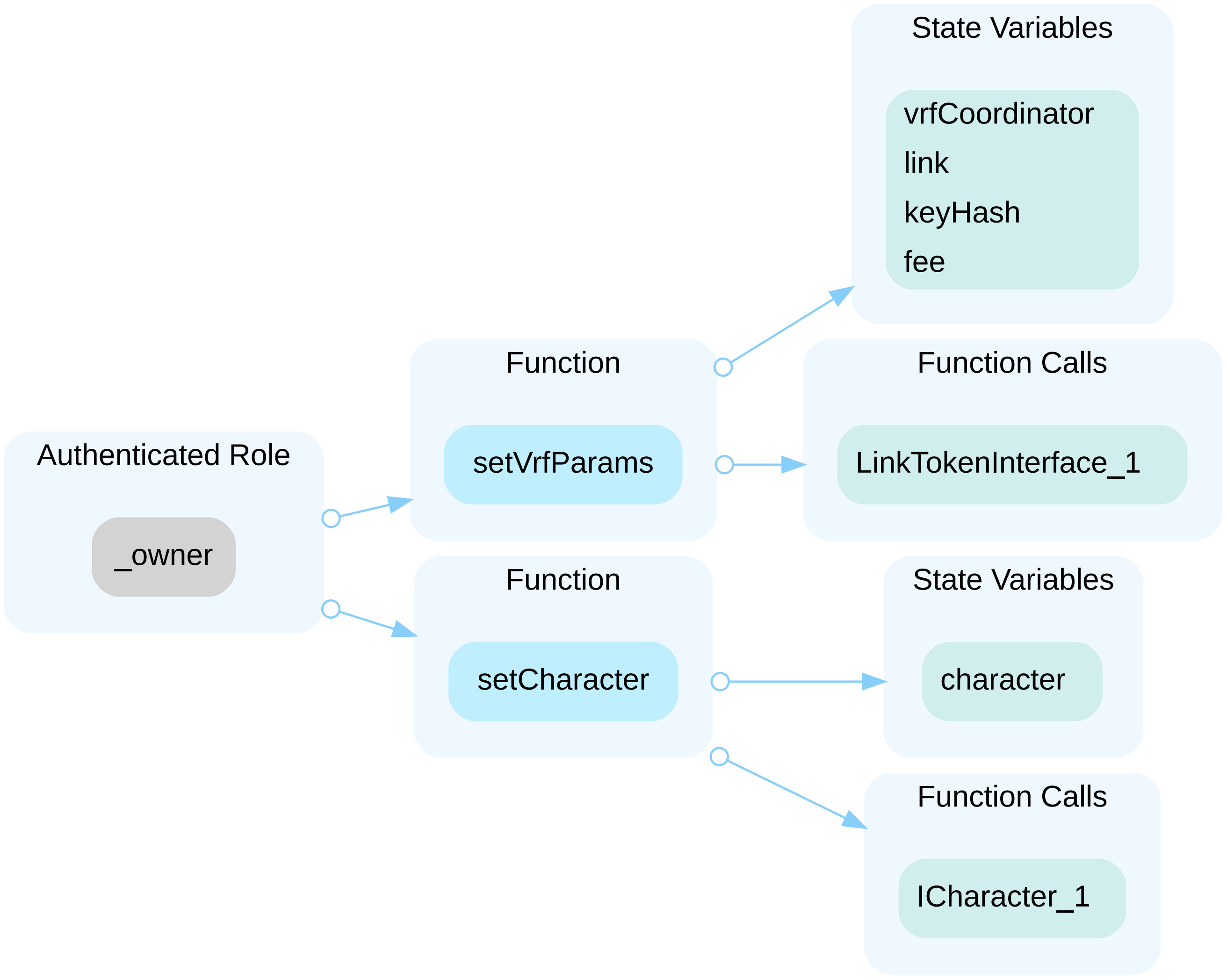
In the contract `Paywall` the role `_owner` has authority over the functions shown in the diagram below.



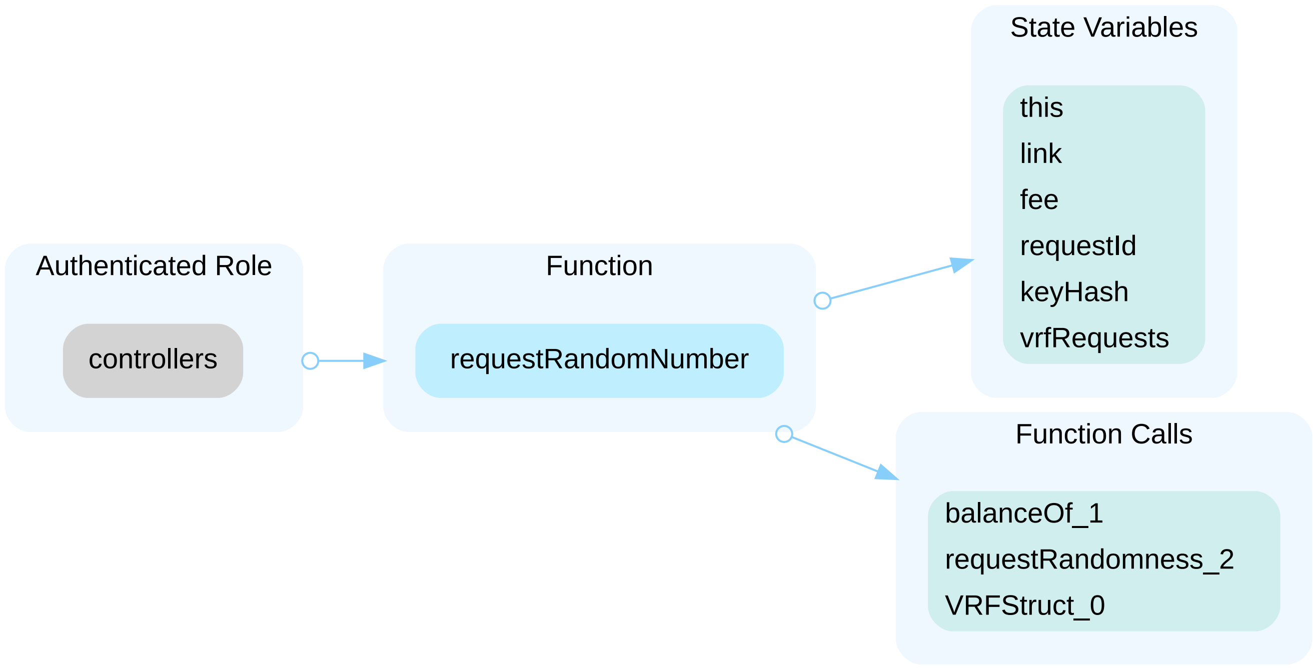
In the contract `Paywall` the role `controllers` has authority over the functions shown in the diagram below.



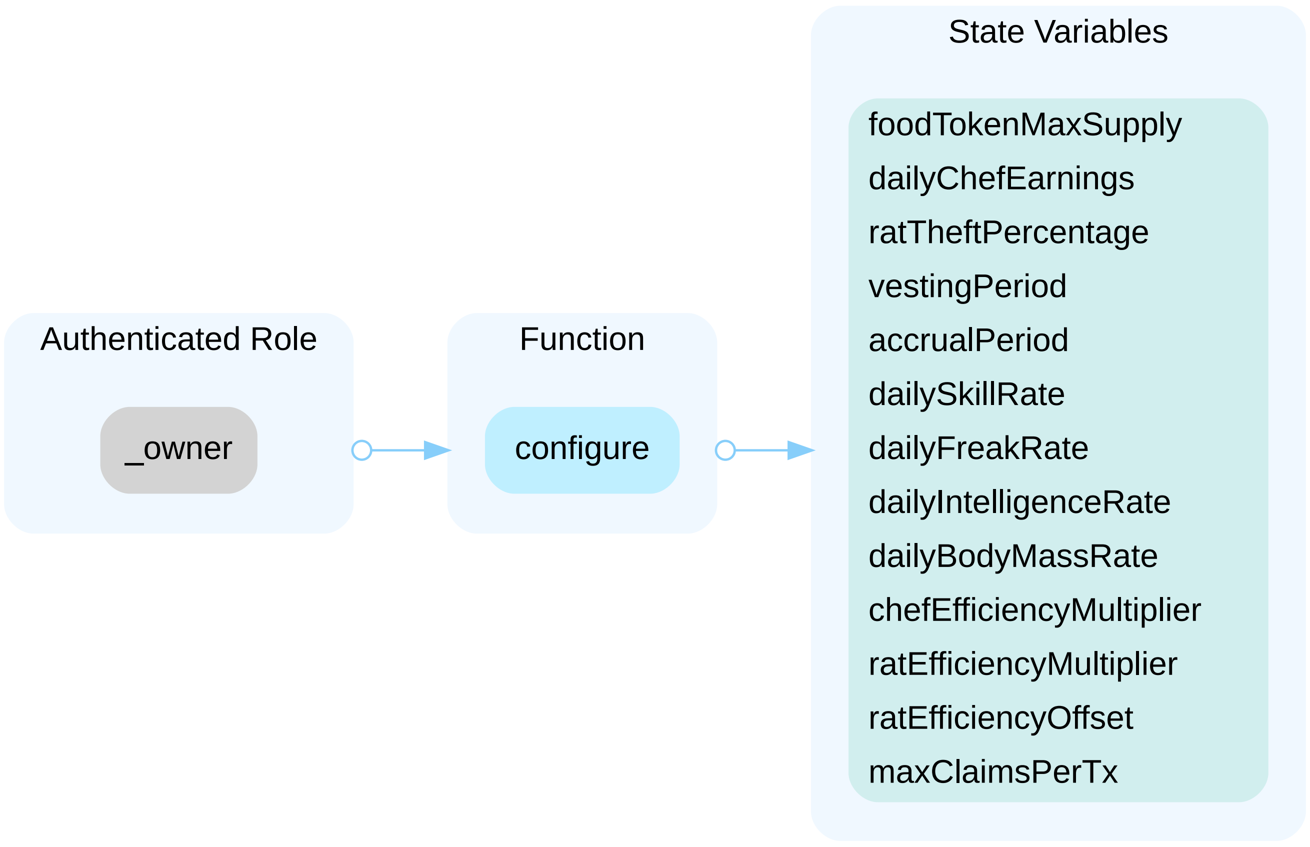
In the contract `Mint` the role `_owner` has authority over the functions shown in the diagram below.



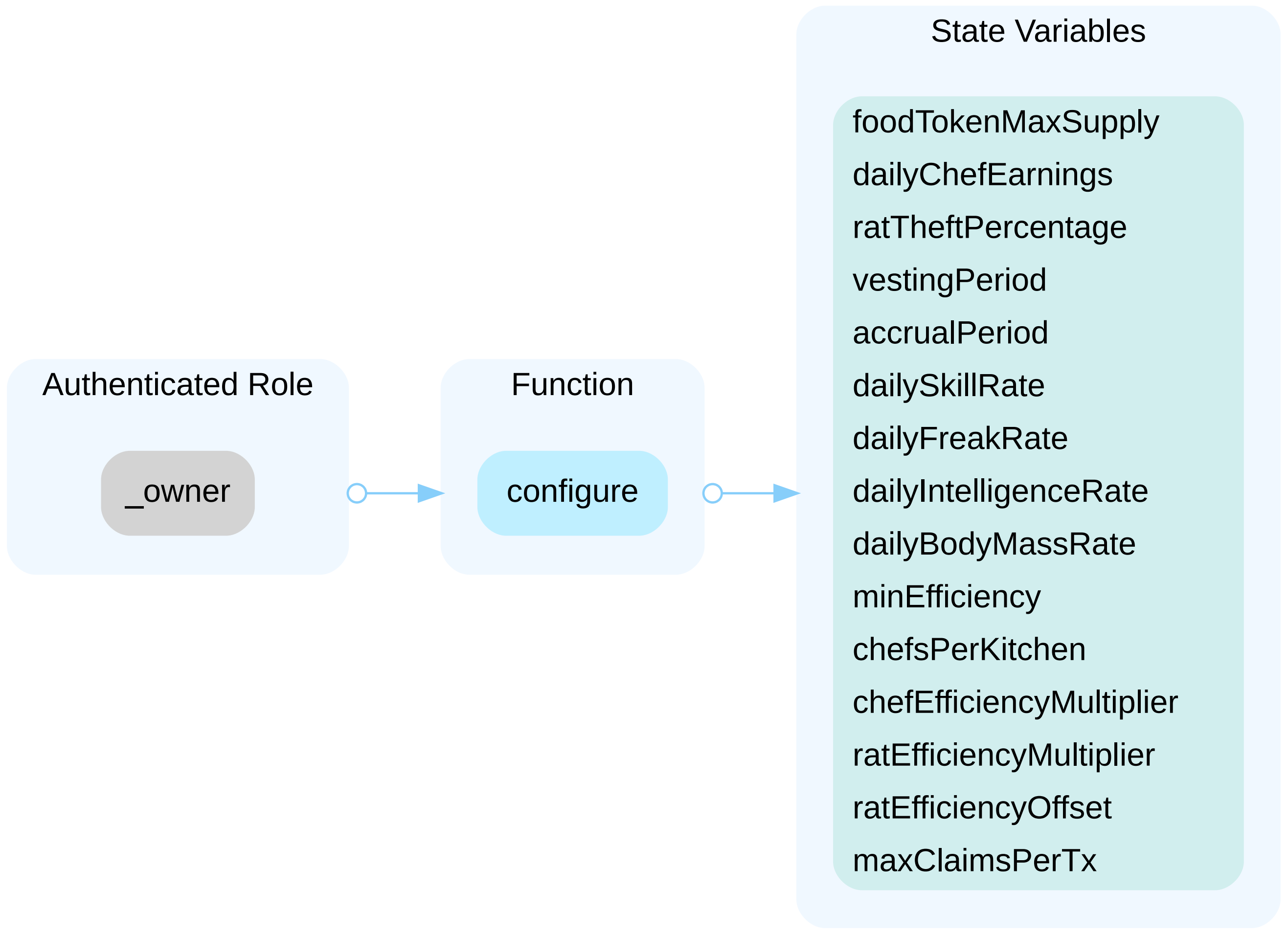
In the contract `Mint` the role `controllers` has authority over the functions shown in the diagram below.



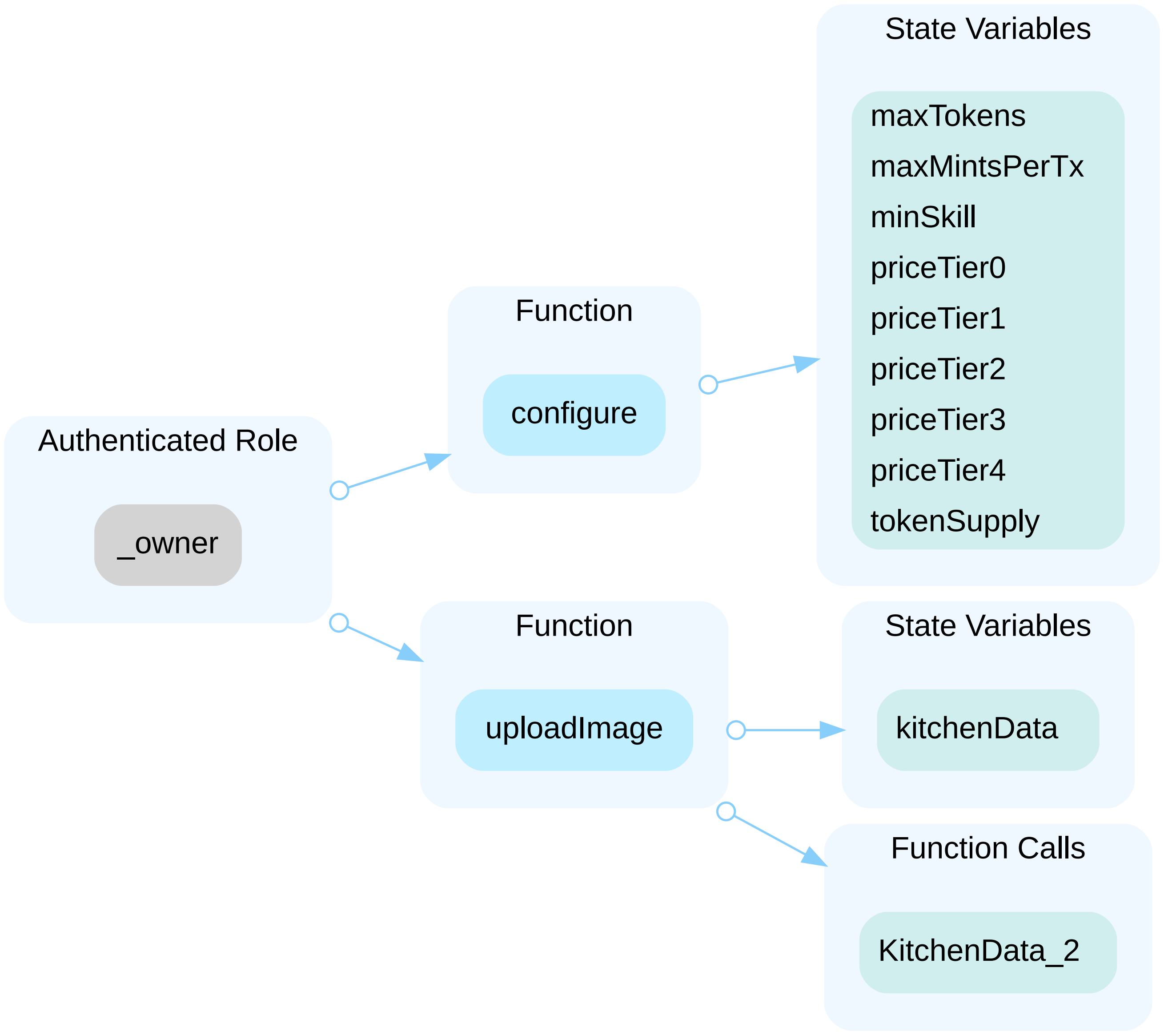
In the contract `McStake` the role `_owner` has authority over the functions shown in the diagram below.



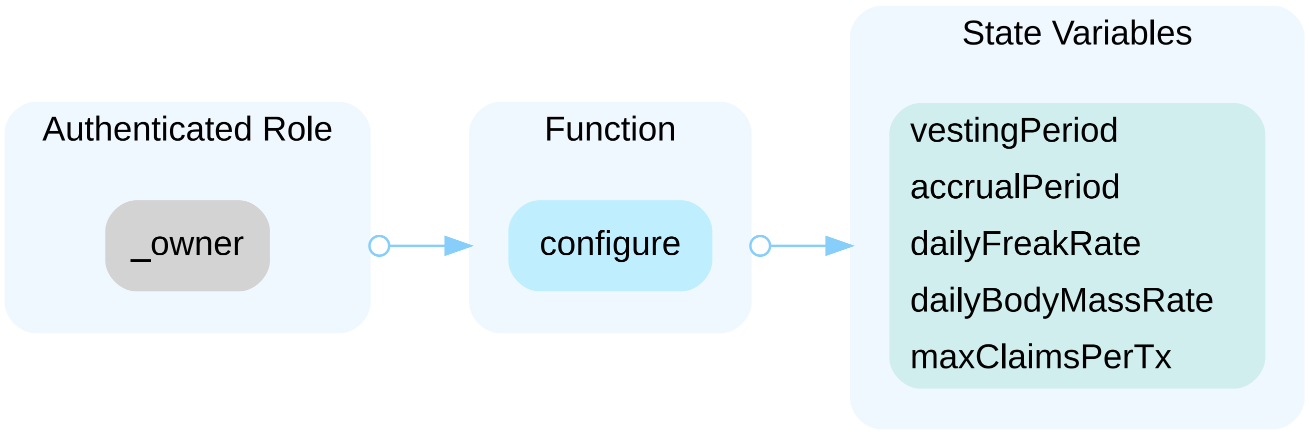
In the contract `LeStake` the role `_owner` has authority over the functions shown in the diagram below.



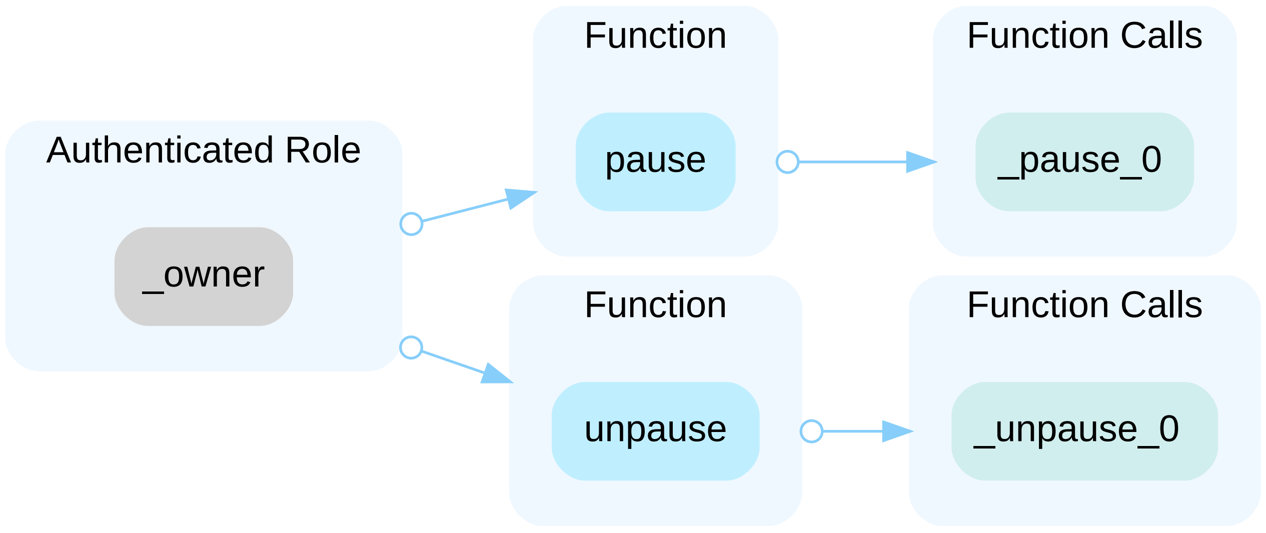
In the contract `KitchenShop` the role `_owner` has authority over the functions shown in the diagram below.



In the contract `Gym` the role `_owner` has authority over the functions shown in the diagram below.



In the contract `GenericPausable` the role `_owner` has authority over the functions shown in the diagram below.



Any compromise to the privileged account may allow the hacker to take advantage of this authority and update the sensitive settings and execute sensitive functions of the project.

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., multi-signature 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 ($2/3$, $3/5$) 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.

Noted: Recommend considering the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

CFB-01 | Inconsistency With White Paper

Category	Severity	Location	Status
Logical Issue	● Minor	contracts/CasualFood.sol: 7~8	✓ Resolved

Description

The white paper describes the supply of CFOOD as **10,000,000**, while in the code this number is **100,000,000** (10 times larger).

```
contract CasualFood is Food {
  constructor() Food("CasualFood", "CFOOD", 100,000,000 * 10 ** 18) {}
}
```

Recommendation

Follow the white paper to implement the code and modify the supply to **10,000,000**

Alleviation

`[CertiK]`: The team heeded the advice and fixed the issue in the commit: 4309f688

CHA-01 | Optimizable Transfer Patterns

Category	Severity	Location	Status
Volatile Code	● Major	contracts/Character.sol: 69~70	✓ Resolved

Description

There is a possibility of exception throwing in dao.transfer, which can make a dos attack.

```
    if (msg.value > 0) {
      dao.transfer(msg.value); // Transfer to Gnosis Safe
    }
```

Specifically, the normal function of character.sol can be affected by dao address.

Exploit

For example, we can make the dao contract unable to accept certain eth transfers, thus preventing the character.sol contract from performing some of the mint operations.

Recommendation

We recommend to use PullPayment model to transfer value.

Alleviation

`[Rat Alert]`: Cannot use OpenZeppelin's PullPayments implementation because it would break the bytecode size limit. Used a custom implementation.

Commit: 8d4c07b5

CHA-02 | Optimizable Data Structure

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/Character.sol: 142~150, 156~161	✓ Resolved

Description

```
  function setVenues(address[] memory _venues) external onlyOwner {
    delete venues;
    for (uint i = 0; i < _venues.length; i++) {
      venues.push(_venues[i]);
    }
  }
```

For the venues in Character, the contract does not use a hashmap to store them, but an array of addresses. This is indeed convenient for off-chain applications to get venues information via eth_getStorageAt()[RPC]. However, this is not in line with the common development model of solidity, and will lead to various serious security problems when the project growth and Owner misuse, as follows.

Possible Impact

The growing venues array will lead to GAS reaching the upper limit causing DOS risk, which may lead to ERC721 tokens not being transferred in RatAlter. And because the venues removal operation is not available, the cost of fixing the problem is extremely high

```
  function transferFrom(address from, address to, uint256 tokenId) public virtual
override {
    bool wl = false;
    for (uint i = 0; i < venues.length; i++) {
      wl = wl || _msgSender() == venues[i];
    }
    if (!wl)
      require(_isApprovedOrOwner(_msgSender(), tokenId), "ERC721: transfer caller is not
owner nor approved");
    _transfer(from, to, tokenId);
  }
```

Recommendation

Using a hashmap to store the "venues" information, you can use events to record the "venues" information that has been stored and the removal to supply the application off-chain.

Alleviation

`[Rat Alert]`: Replaced the entire venues array with a single venue address.

Commit: 10d55080

CLA-01 | Users Do Not Pay For $link Usage

Category	Severity	Location	Status
Logical Issue	● Minor	contracts/Claim.sol: 61~62	✓ Resolved

Description

Users only need to use the native token mint ERC721 tokens at the beginning of the game and pay nothing beyond the transaction fee after that.

```
function requestRandomness(bytes32 _keyHash, uint256 _fee) internal returns
(bytes32 requestId) {
    link.transferAndCall(vrfCoordinator, _fee, abi.encode(_keyHash,
USER_SEED_PLACEHOLDER));
    uint256 vRFSeed = makeVRFInputSeed(_keyHash, USER_SEED_PLACEHOLDER,
address(this), nonces[_keyHash]);
    nonces[_keyHash] = nonces[_keyHash] + 1;
    return makeRequestId(_keyHash, vRFSeed);
}
```

However, any call to `requestRandomNumber()` consumes link tokens, which gives malicious users the possibility to attack the project owner.

A malicious user can consume the project owner's link tokens by using FFOOD tokens to generate ERC721 tokens or by pledging ERC721 tokens. And in the process, they can also harvest more FOOD tokens without paying anything other than transaction fees.

A malicious user may benefit from this process as follows.

1. DOS contracts that prevent them from performing claim operations while increasing the benefit of their own claims.
2. consume the project's LINK tokens, causing financial losses to the project.

Recommendation

Please consider a mechanism where the link tokens are paid by the user, or guarantee that the user has to pay a suitable price to consume the link tokens.

Alleviation

`[Rat Alert]`: Vendor.claimMany() now requires a a claim fee. However, skipping Character.mint() since it requires a payment anyway.

Commit: 64f20f98

CON-01 | Flawed $Kitchen Token Implementation

Category	Severity	Location	Status
Logical Issue	● Medium	contracts/Venue.sol: 77~80, 149~155; contracts/EntrepreneurKitchen.sol: 19 ~22	✓ Resolved

Description

The kitchen token is a credential provided to chefs to access LeStake as well as TheStakeHouse. However, these tokens are not stake in contracts when it is being used. It is only to check the user balance at the time of stakeChef and claimChef. This leads to the following unintended actions that can be made.

1. User A mint a certain amount of $kitchen tokens first, and does a stakeChef operation.
2. User A transfers $kitchen to B
3. B performs the stakechef operation, and then transfers $kitchen to A
4. A performs claimchef operation in the future and transfers $kitchen to B
5. B performs the claimchef operation

In this way, a $Kitchen is used twice in a time period, ultimately increasing the revenue of $XFOOD tokens

Recommendation

Stake $kitchen tokens in contracts such as Lestake when using it.

Alleviation

`[Rat Alert]`: Kitchens now require approval and staking in the KitchenUsage contract (ERC1155 receiver) as long as chefs are staked.

Commit: 4493b12c

CON-02 | Potential Reentrancy Attack

Category	Severity	Location	Status
Logical Issue	● Minor	contracts/KitchenShop.sol: 7~8, 95~99; contracts/Character.sol: 6~7, 88~89; contracts/Venue.sol: 71~72, 190~191, 225~226	✓ 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.

And there are numerous `checkReceived` checks in the ERC721 contract as well as in the ERC1155 contract, which can lead to re-entry problems and eventually to unpredictable logic errors.

```
require(tx.origin == _msgSender(), "EOA only");
```

The use of msg.sender==tx.orign does prevent some reentrant attacks, but we also see many places where this check is not present and can certainly cause reentrants, especially in the context of many functions that do not adhere to CEI principles.

```
    if (unstake) {
      character.safeTransferFrom(address(this), sender, tokenId, ""); // Send Chef back
to owner
      delete chefs[tokenId];
      totalChefsStaked --;
    }
```

Reentrant attacks can take many forms, and it should to eliminate the possibility of reentrants in all functions, even if they sometimes do not appear to be harmful at first glance.

Recommendation

We recommend

1.using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts

2.applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

`[Rat Alert]`:

- Contracts have been refactored to adhere to CEI principles.
- Introduced "EOA only" checks in those functions that users interact with. All other functions either require onlyOwner, onlyDAO or onlyController.
- Aside from ChainLink VRM, all contracts are owned and can be considered trusted.

Commit: a013b732

CON-03 | Optimizable Emit Events Design

Category	Severity	Location	Status
Coding Style	● Minor	contracts/KitchenShop.sol: 57~74, 158~163; contracts/Mint.sol: 63~68, 193~195; contracts/Traits.sol: 196~198, 205~213; contracts/Character.sol: 54~57, 103~110, 156~161; contracts/Food.sol: 25~27, 34~36; contracts/ControllableUpgradeable.sol: 22~24, 30~34, 40~44; contracts/VRFConsumer.sol: 51~54, 61~63; contracts/Gym.sol: 22~34; contracts/Migrations.sol: 16~18; contracts/GenericPausable.sol: 9, 10; contracts/Claim.sol: 47~52, 86~90, 96~100; contracts/McStake.sol: 31~53; contracts/Controllable.sol: 7~8, 22~24, 30~34, 40~44; contracts/LeStake.sol: 35~61; contracts/Properties.sol: 31~45; contracts/Paywall.sol: 36~44, 50~52; contracts/TheStakehouse.sol: 35~61	ⓘ Acknowledged

Description

Events are an important implementation for Ether to provide contract operation info to the off-chain and provide data for the off-chain monitoring facility.Events should be designed to ensure coverage enough as well as difficult collisions.

But in the rat alter we found the following problems with the event design:

1. The function that affects the status of sensitive variables should be able to emit events as notifications to all of the function of [Controllable.sol] [Character.sol].

- `addController()`
- `removeController()`
- `setVenues()` ...

2. The output event information should give a more complete picture of the chain, There should be no ambiguity about different executions but outputting the same event. That is, a collision occurs

```
  function _stakeRat(address account, uint256 tokenId) internal whenNotPaused {
    ...
    emit TokenStaked(tokenId, account, foodTokensPerRat);//Can't tell if it's chefs or
rats.
  }
  function _stakeChef(address account, uint256 tokenId) internal whenNotPaused {
    ...
    emit TokenStaked(tokenId, account, block.timestamp);//Can't tell if it's chefs or
rats.
  }
```

Recommendation

Consider adding events for important actions, and emit them in the function, and ensure that the meaning of the event is clear.

Alleviation

`[Rat Alert]`: Character & contracts that inherit from Venue are close to the bytecode size limit. Although we would love to add a lot more events, we need to stick to only a few that are critical for operation.

CON-04 | Receive Token By `burn()`

Category	Severity	Location	Status
Logical Issue	● Minor	contracts/KitchenShop.sol: 95~99; contracts/Paywall.sol: 141~142	Partially Resolved

Description

In the following code snippet:

```
  function handle(address sender, uint8 amount, uint256 msgValue, uint16 minted, uint256
maxTokens, uint256 gen0Tokens) external onlyController returns (int8 boost) {
    ...
    if (totalCost > 0) fastFood.burn(sender, totalCost);
  }
```

It is very unreasonable to use burn mechanism in a Receive token function that has ERC20Capped. It would cause the following security risk.

1. **causing a severe shortage of token supply.** It may cause the project to run out of FFood or CFood during its lifetime, causing some of the project's functions to fail, such as the mint operation of ERC721 tokens.
2. **Amplifies the impact of centralization issues.** The burn mechanism is a high authority operation that does not require the authority of the authenticated token owner. This operation is given to all Controllers, which also contain Upgradeable contracts [Paywall, LeStake, etc.], and the Upgradability of these contracts may further aggravate the centralization problem of the project.

Recommendation

It is recommended to use the normal token transfer mechanism to receive ERC20 tokens, e.g. **safetransferfrom()**

Alleviation

`[Rat Alert]`: Replaced Controllable with OpenZeppelin's AccessControl to better protect the ERC20 contracts.

Commit: 7cc6e807

CON-05 | Bad ERC721 Token Stake Method

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/Venue.sol: 69~74; contracts/Character.sol: 142~150	✓ Resolved

Description

In the venues contract running the ERC721 token transfer, token owner authentication is implemented by venues contract, and for ERC721 contract it is not appropriate to skip the _isApprovedOrOwner() check for transactions sent from venues. This implementation exacerbates the project coupling level and makes the project security risk higher. This increases the degree of project centrality.

Recommendation

Please use _isApprovedOrOwner() check for venues as well, and try to have the user provide ERC721 to venues via approve first, and then have the contract transfer it later

Alleviation

`[Rat Alert]`: Got it. All contracts now require approval.

Commit: 81dec95f

CON-06 | Payment Token By Mint()

Category	Severity	Location	Status
Logical Issue	● Minor	contracts/TheStakehouse.sol: 68~70; contracts/McStake.sol: 60~62; contracts/LeStake.sol: 68~70	Partially Resolved

Description

```
function _mintFoodToken(address sender, uint256 amount) internal override {
  foodToken.mint(sender, amount);
}
```

It is dangerous to use mint mechanism in a Payment function. It may cause the following results.

1. **The risk of breaking the economic model of the FOOD token as described in the white paper**. This may further result in users not being able to claim their ERC721token.

2. **Amplifies the impact of centralization issues.** The mint mechanism is a high authority operation that does not require the authority of the authenticated token owner. This operation is given to all Controllers, which also contain Upgradeable contracts, and the Upgradability of these contracts may further aggravate the centralization problem of the project.

Recommendation

It is recommended to mint enough FOOD tokens in the initial phase of the project and distribute them according to the economic model, and use **safeTransferFrom()** to make payments.

It is also recommended to add some simple ERC721 withdrawal mechanism to prevent the problem of permanent ERC721 staking.

Alleviation

`[Rat Alert]`: Replaced Controllable with OpenZeppelin's AccessControl to better protect the ERC20 contracts.

Commit: 7cc6e807

CON-07 | Risk Of Transaction Revert Due To Gas Reaching Limit

Category	Severity	Location	Status
Logical Issue	● Minor	contracts/Character.sol: 65~94; contracts/Venue.sol: 141~166	✓ Resolved

Description

Be aware of the risk of fullfillClaimMany reaching the gas limit due to too many tokenIDs, which will lead to revert of fullfillClaimMany transactions. The same problem can occur in all functions that call requestRandomNumber such as mint() in Character.

```
  function mint(uint8 amount, bool stake) external payable whenNotPaused {
    ...
    theMint.requestRandomNumber(_msgSender(), amount, stake, boost);
  }
```

```
  function claimMany(uint16[] calldata tokenIds, bool unstake) external virtual payable
whenNotPaused {
    ...
    bytes32 requestId = claim.requestRandomNumber(_msgSender(), tokenIds, unstake);
    ...
  }
  function fulfillClaimMany(IClaim.VRFStruct memory v, uint256 randomness) external
virtual whenNotPaused {
    //spend many gas here
    require(msg.sender == address(claim), "Only Claim can fulfill");
    require(claimRequests[v.requestId].length > 0, "Claim request not found");

    uint16[] memory tokenIds = claimRequests[v.requestId];
    delete claimRequests[v.requestId];

    uint256 owed = 0;
    for (uint i = 0; i < tokenIds.length; i++) {
      uint256 randomVal = uint256(keccak256(abi.encode(randomness, i)));
      bool space = _checkSpace(v.sender, 0);
      if (isChef(tokenIds[i]))
        owed += _claimChef(tokenIds[i], v.sender, !space || v.unstake, !space,
randomVal);
      else
        owed += _claimRat(tokenIds[i], v.sender, v.unstake, !space, randomVal);
      for (uint j = 0; j < stakers[v.sender].length; j++) {
        if (stakers[v.sender][j] == tokenIds[i]) {
          stakers[v.sender][j] = stakers[v.sender][stakers[v.sender].length - 1];
          stakers[v.sender].pop();
        }
      }
    }
    if (owed > 0) {
      _mintFoodToken(v.sender, owed);
    }
  }
```

Recommendation

It is recommended to check the number of tokenids on the front-end and to warn users in the website not to provide more than the recommended number of tokenids at once.

Alleviation

`[Rat Alert]`: we noticed that already and did one better: Both Venue.claimMany() and Character.mint() via Paywall.handle() ensure that the limit is not violated.

CON-08 | Improper Usage Of `public` And `external` Type

Category	Severity	Location	Status
Gas Optimization	● Informational	contracts/KitchenShop.sol: 122; contracts/Traits.sol: 55; contracts/Character.sol: 103, 142; contracts/Venue.sol: 342; contracts/Migrations.sol: 16; contracts/Properties.sol: 128	Partially Resolved

Description

`public` functions that are never called by the contract could be declared as `external`. `external` functions are more efficient than `public` functions.

Recommendation

Consider using the external attribute for public functions that are never called within the contract.

Alleviation

`[Rat Alert]`: Cannot change the following:

- Character.transferFrom() because it overrides an OpenZeppelin function
- KitchenShop.uri() because it overrides an OpenZeppelin function

Changed the remaining ones:

- Character.updateCharacter()
- Migrations.setCompleted()
- Properties.getEventUpdates()
- Traits.tokenURI()
- Venue.getProperties()

Commit: bb104ea2 & cd2bfdf7

CON-09 | Using Standardized Base64 Library

Category	Severity	Location	Status
Coding Style	● Informational	contracts/KitchenShop.sol: 169; contracts/Traits.sol: 218~251	✓ Resolved

Description

Traits is using the Base64 library implementation written by Brech Devos, which is undoubtedly a widely used and borrowed library, but as of 14 Sep 2021 Openzeppelin has implemented a base64 library code by borrowing from Brech Devos. It was equipped with more standardized documentation and testing efforts, as well as some optimizations. For better maintenance and modularity later in the project. It is recommended to use Openzeppelin's Base64 library, as it has a larger community to maintain it and is equipped with better safeguards such as documentation and bug bounties.

Recommendation

Use the base64 library code of openzeppelin.

Alleviation

`[Rat Alert]`: Now using the OpenZeppelin's base64 implementation, had to upgrade @openzeppelin/contracts & @openzeppelin/contracts-upgradeable.

Commit: 0b379a42

Appendix

Finding Categories

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.

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.

Disclaimer

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.

About

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.