Aavegotchi

Smart Contracts

Security Assessment

March 1st, 2021

Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are 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 review.

CertiK Reports do 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.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing 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.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	Aavegotchi Smart Contracts
Description	Smart contracts portion of the Aavegotchi repository
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. 7376afa4ef247bd7f78fe4f6c3643d3fac9008f2 2. 6536d434b7c87c1cf0325917f811bf042e74ef50

Audit Summary

Delivery Date	March. 1, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	3
Timeline	Dec. 21, 2020 - March. 1, 2021

Vulnerability Summary

Total Issues	41
Total Critical	0
Total Major	1
Total Medium	2
Total Minor	7
Total Informational	31

Executive Summary

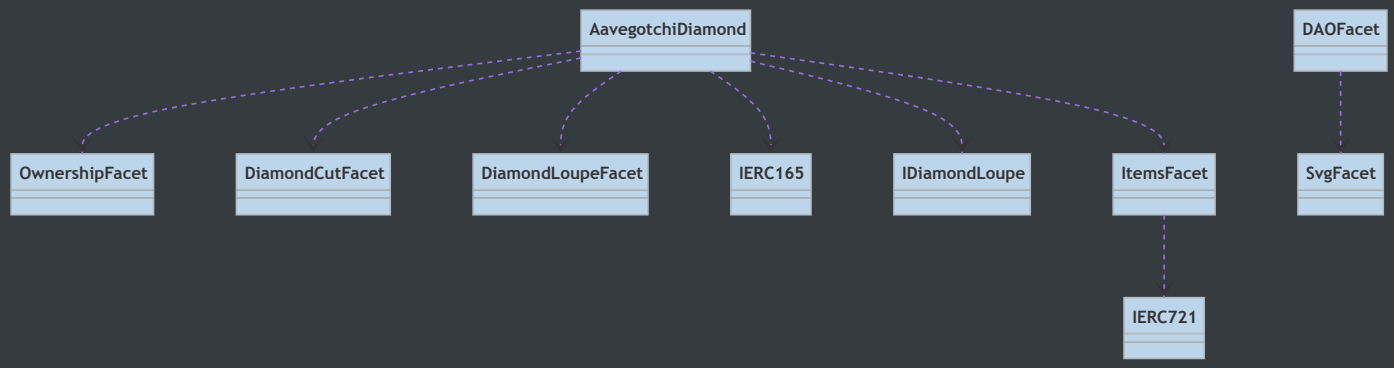
This report represents the results of CertiK's engagement with Aavegotchi on their implementation of the Aavegotchi smart contracts.

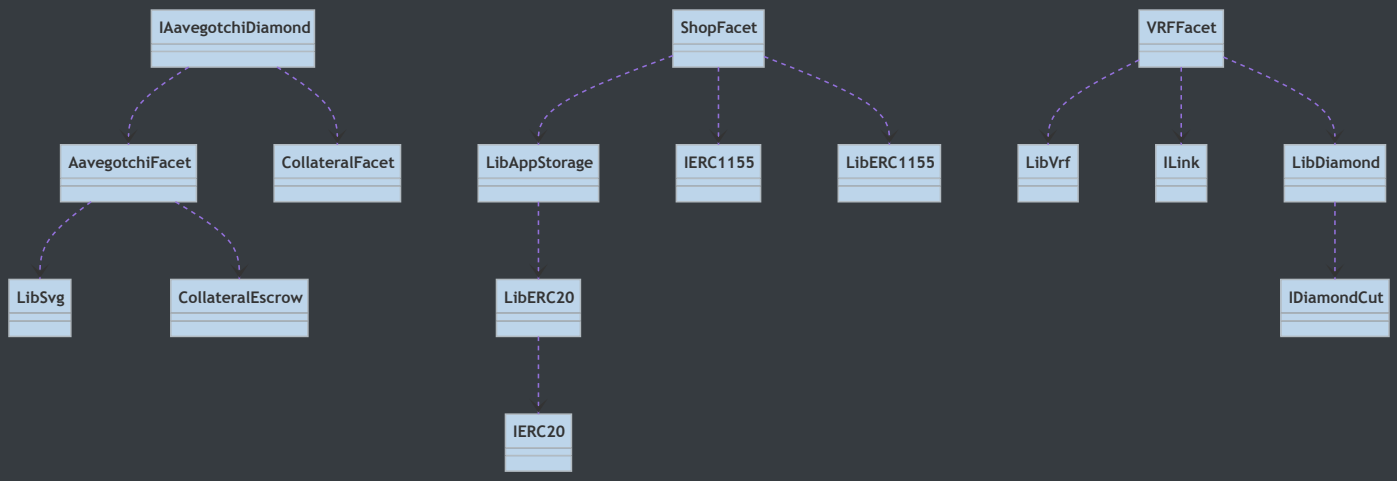
Static analysis and manual inspection was performed on the smart contracts in scope. Most of the findings are of informational nature with a few medium and minor findings. Majority of the issues were remediated except a couple of informational findings, which were not considered.

Files In Scope

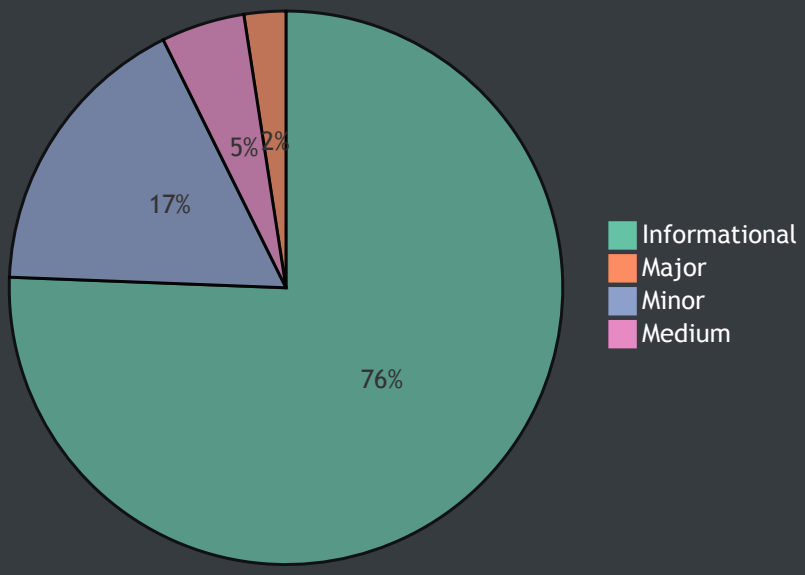
ID	Contract	Location
AFT	AavegotchiFacet.sol	contracts/Aavegotchi/facets/AavegotchiFacet.sol
ADD	AavegotchiDiamond.sol	contracts/Aavegotchi/AavegotchiDiamond.sol
CFT	CollateralFacet.sol	contracts/Aavegotchi/facets/CollateralFacet.sol
CEW	CollateralEscrow.sol	contracts/Aavegotchi/CollateralEscrow.sol
DAO	DAOFacet.sol	contracts/Aavegotchi/facets/DAOFacet.sol
ILK	ILink.sol	contracts/Aavegotchi/interfaces/ILink.sol
IER	IERC721.sol	contracts/Aavegotchi/interfaces/IERC721.sol
IEC	IERC1155.sol	contracts/Aavegotchi/interfaces/IERC1155.sol
IFT	ItemsFacet.sol	contracts/Aavegotchi/facets/ItemsFacet.sol
IAD	IAavegotchiDiamond.sol	contracts/Aavegotchi/interfaces/IAavegotchiDiamond.sol
IET	IERC1155TokenReceiver.sol	contracts/Aavegotchi/interfaces/IERC1155TokenReceiver.sol
LVF	LibVrf.sol	contracts/Aavegotchi/libraries/LibVrf.sol
LER	LibERC20.sol	contracts/shared/libraries/LibERC20.sol
LDD	LibDiamond.sol	contracts/shared/libraries/LibDiamond.sol
LEC	LibERC1155.sol	contracts/Aavegotchi/libraries/LibERC1155.sol
LAS	LibAppStorage.sol	contracts/Aavegotchi/libraries/LibAppStorage.sol
SFT	ShopFacet.sol	contracts/Aavegotchi/facets/ShopFacet.sol
VRF	VRFFacet.sol	contracts/Aavegotchi/facets/VRFFacet.sol

File Dependency Graph





Findings



ID	Title	Type	Severity	Resolved
LAS-01	Inefficient storage read	Gas Optimization	Informational	✓
LAS-02	Inefficient storage layout	Gas Optimization	Informational	✓
LAS-03	Redundant Statements	Dead Code	Informational	✓
LAS-04	Unsafe subtraction	Arithmetic	Minor	✓
AFT-01	Documentation discrepancy	Inconsistency	Informational	✓
AFT-02	Inefficient Greater-Than Comparison w/ Zero	Gas Optimization	Informational	✓
AFT-03	Code readability can be improved	Coding Style	Informational	✓
AFT-04	Code readability can be improved	Coding Style	Informational	✓

AFT-05	Array type can be changed from dynamic to fixed length	Gas Optimization	Informational	✓
AFT-06	Inefficient code	Gas Optimization	Informational	✓
AFT-07	Redundant `return` statement	Coding Style	Informational	✓
AFT-08	Inefficient local variable	Gas Optimization	Informational	✓
AFT-09	Documentation discrepancy	Inconsistency	Informational	✓
AFT-10	Comparison with literal `false`	Gas Optimization	Informational	✓
AFT-11	Redundant Variable Initialization	Coding Style	Informational	✓
AFT-12	`uint8` type can be changed to `uint256`	Gas Optimization	Informational	✓
AFT-13	`require` statements can be substituted with `modifier`	Gas Optimization	Informational	!
AFT-14	Documentation discrepancy	Logical Issue	Major	✓
CFT-01	Duplicate code	Coding Style	Informational	✓
CFT-02	Possibility of integer underflow	Arithmetic	Minor	✓
CFT-03	Incorrect word	Coding Style	Informational	✓
DAO-01	`emit` keyword is missing before the event call	Language Specific	Minor	✓
DAO-02	Possibility of integer overflow	Arithmetic	Minor	✓
SFT-01	Redundant storage read	Gas Optimization	Informational	✓
SFT-02	Possibility of integer overflow	Arithmetic	Minor	✓
SFT-03	Possibility of integer overflow	Arithmetic	Minor	✓
SFT-04	Inefficient Greater-Than Comparison w/ Zero	Gas Optimization	Informational	✓
VRF-01	Comparison with literal `false`	Gas Optimization	Informational	✓
VRF-02	Comparison with literal `true`	Gas Optimization	Informational	✓
VRF-03	Returned `success` value is not checked of function call	Volatile Code	Minor	✓

VRF-04	`require` statement can be substituted with `modifier`	Gas Optimization	Informational	✓
IFT-01	Incorrect code	Control Flow	Medium	✓
IFT-02	`uint256` can be used instead of `uint16`	Gas Optimization	Informational	✓
IFT-03	Inefficient function implementation	Gas Optimization	Informational	✓
IFT-04	Comparison with a literal `true`	Gas Optimization	Informational	✓
IFT-05	Duplicate code can be extracted to a function	Gas Optimization	Informational	✓
IFT-06	Duplicate code can be extracted to a function	Gas Optimization	Informational	✓
ADD-01	Ether locking in AavegotchiDiamond	Volatile Code	Medium	✓
LDD-01	Inefficient storage struct layout	Gas Optimization	Informational	✓
LDD-02	Redundant `require` statement	Gas Optimization	Informational	✓
LDD-03	Inefficient Greater-Than Comparison w/ Zero	Gas Optimization	Informational	✓

LAS-01: Inefficient storage read

Type	Severity	Location
Gas Optimization	Informational	LibAppStorage.sol L164-L167

Description:

The aforementioned lines read storage variable `s.ghstContract` multiple times. Reading from storage is significantly expensive than reading from a local variable and to enhance the gas efficiency of code, we suggest that the storage variable is first stored in a local variable and then used on the aforementioned lines.

Recommendation:

We recommend to store the storage variable of `s.ghstContract` into a local variable and then use it on the aforementioned lines.

```
    address ghstAddress = s.ghstContract;
    LibERC20.transferFrom(s.ghstContract, msg.sender,
    address(0xFFfFfFffFFfffFFfFFfFFFFFffFFFffffFfFFFfF), burnShare);
    LibERC20.transferFrom(ghstContract, msg.sender, s.pixelCraft, companyShare);
    LibERC20.transferFrom(ghstContract, msg.sender, s.rarityFarming, rarityFarmShare);
    LibERC20.transferFrom(ghstContract, msg.sender, s.dao, daoShare);
```

Alleviation:

Alleviations were applied as of commit hash `6536d434b7c87c1cf0325917f811bf042e74ef50`.

LAS-02: Inefficient storage layout

Type	Severity	Location
Gas Optimization	Informational	LibAppStorage.sol L102-L103

Description:

The struct `AppStorage` 's properties on the aforementioned lines are laid out in an inefficient way where they occupy the a complete 32-byte slot. These properties can be placed at the end of the struct where they will be packed with the `address` type property resulting in saving gas cost associated with an additional storage slot.

Recommendation:

We advise to place the struct properties on the aforementioned lines at the end of the struct so they can be tight packed with the `address` type.

```
struct AppStorage {
    ...
    address rarityFarming;
    uint32 totalSupply;
    uint16 currentHauntId;
}
```

Alleviation:

The recommendation was not considered, with the Aavegotchi team stating "Understood. Since moving to Polygon (which has very low gas fees) we are less concerned with inefficient storage layout and more concerned with convenience and contract readability. Therefore I did not change this item".

LAS-03: Redundant Statements

Type	Severity	Location
Dead Code	Informational	LibAppStorage.sol L170-L175

Description:

The linked statements do not affect the functionality of the codebase and appear to be either leftovers from test code or older functionality.

Recommendation:

We advise that they are removed to better prepare the code for production environments.

Alleviation:

Alleviations were applied as of commit hash `6536d434b7c87c1cf0325917f811bf042e74ef50`.

LAS-04: Unsafe subtraction

Type	Severity	Location
Arithmetic	Minor	LibAppStorage.sol L221

Description:

The aforementioned line performs unsafe subtraction which can result in underflow of the resultant value.

Recommendation:

We recommend to either use `SafeMath` library from OpenZeppelin or introduce a check asserting that the minuend is larger than or equal to the subtrahend.

```
require(
   _experience >= _lowRange,
   "_experience is smaller than _lowRange"
);
```

For `SafeMath` library, the following link can be reffered.

```
https://github.com/OpenZeppelin/openzeppelin-
contracts/blob/master/contracts/math/SafeMath.sol
```

Alleviation:

Alleviations were applied as of commit hash `6536d434b7c87c1cf0325917f811bf042e74ef50` by removing the relevant code part.

AFT-01: Documentation discrepancy

Type	Severity	Location
Inconsistency	Informational	AavegotchiFacet.sol L156-L163

Description:

The comments on the aforementioned lines suggest that the function `balanceOf` should throw if it is called with the argument of zero address yet the function does not implement logic to throw on such event.

Recommendation:

We advise to either implement the logic in `balanceOf` function which throws upon `_owner` being passed as `address(0)` or remove the comments expecting this behaviour from the function to increase the legibility of the codebase.

Alleviation:

Alleviations were applied as of commit hash `6536d434b7c87c1cf0325917f811bf042e74ef50`.

AFT-02: Inefficient Greater-Than Comparison w/ Zero

Type	Severity	Location
Gas Optimization	Informational	AavegotchiFacet.sol L196

Description:

The linked greater-than comparisons with zero compare variables that are restrained to the non-negative integer range, meaning that the comparator can be changed to an inequality one which is more gas efficient.

Recommendation:

We advise that the above paradigm is applied to the linked greater-than statements.

Alleviation:

The finding was incorrectly identified as the value being compared with zero can be a negative number. This exhibit is rendered ineffective.

AFT-03: Code readability can be improved

Type	Severity	Location
Coding Style	Informational	AavegotchiFacet.sol L196-L199

Description:

The `if` statements on the aforementioned lines can be combined in a single `if` statement to increase readability of the codebase.

Recommendation:

We recommend to combine the `if` statements on the aforementioned lines to increase the readability of the codebase.

```
if (boost > 0 && boost > boostDecay) {...}
```

Alleviation:

Alleviations were applied as of commit hash `6536d434b7c87c1cf0325917f811bf042e74ef50`.

AFT-04: Code readability can be improved

Type	Severity	Location
Coding Style	Informational	AavegotchiFacet.sol L200-L204

Description:

The aforementioned lines has `else` block and an `if` block inside it. These blocks can be combined to increase the readability of the codebase.

Recommendation:

We advise to combine the `else` and `if` block to increase the readability of the codebase.

```
if () {}
else if ((boost * -1) > boostDecay) {
   number += boost + boostDecay;
}
```

Alleviation:

Alleviations were applied as of commit hash `6536d434b7c87c1cf0325917f811bf042e74ef50` .

AFT-05: Array type can be changed from dynamic to fixed length

Type	Severity	Location
Gas Optimization	Informational	AavegotchiFacet.sol L171, L216

Description:

The aforementioned lines declare dynamic arrays as part of struct properties. The initialization of these arrays in the code always result in a fixed length equal to `LibAppStorage.NUMERIC_TRAITS_NUM` and hence the type of these arrays can be changed from dynamic to fixed-length array with length being `LibAppStorage.NUMERIC_TRAITS_NUM`.

Recommendation:

We advise to change the type of arrays on the aforementioned lines from dynamic to fixed-length array where length is equal to `LibAppStorage.NUMERIC_TRAITS_NUM`.

```
    int256[LibAppStorage.NUMERIC_TRAITS_NUM] numericTraits;
```

Alleviation:

Alleviations were applied as of commit hash `6536d434b7c87c1cf0325917f811bf042e74ef50`.

AFT-06: Inefficient code

Type	Severity	Location
Gas Optimization	Informational	AavegotchiFacet.sol L227-L231

Description:

The code on the aforementioned lines can be optimized by replacing the `if-else` statements with only an `if` block which sets `stakeAmount` when `collateral` is a non-zero address.

Recommendation:

We advise to replace the `if-else` statements with an `if` block which sets `stakeAmount` when `collateral` is a non-zero address. Setting of `stakeAmount` to `0` is not needed as it is the default value of `uint256` type.

```
    if (aavegotchiInfo_.collateral != address(0)) {
       aavegotchiInfo_.stakedAmount =
    IERC20(aavegotchiInfo_.collateral).balanceOf(aavegotchiInfo_.escrow);
    }
```

Alleviation:

Alleviations were applied as of commit hash `6536d434b7c87c1cf0325917f811bf042e74ef50`.

AFT-07: Redundant `return` statement

Type	Severity	Location
Coding Style	Informational	AavegotchiFacet.sol L240

Description:

The `return` statement on the aforementioned line is redundant as the function `getAavegotchi` has a named return parameter `aavegotchiInfo_` which is implicitly returned at the end of function execution and hence explicit return statement can be removed.

Recommendation:

We recommend to remove the explicit `return` statement at the end of aforementioned function's body.

Alleviation:

Alleviations were applied as of commit hash `6536d434b7c87c1cf0325917f811bf042e74ef50`.

AFT-08: Inefficient local variable

Type	Severity	Location
Gas Optimization	Informational	AavegotchiFacet.sol L270-L271

Description:

The local variable on the aforementioned line copies the struct from storage to memory and is only utilized once in the code. This is an inefficient implementation as simply reading a value from storage will be significantly cheaper than first copying the whole struct in memory and then reading a value from it.

Recommendation:

We recommend to remove the local variable `collateralInfo` and directly read the value from storage on `L271`.

```
    uint256 modifiers = s.collateralTypeInfo[collateralType].modifiers;
```

Alleviation:

Alleviations were applied as of commit hash `6536d434b7c87c1cf0325917f811bf042e74ef50`.

AFT-09: Documentation discrepancy

Type	Severity	Location
Inconsistency	Informational	AavegotchiFacet.sol L363-L370

Description:

The comments on the aforementioned lines suggest that the function `ownerOf` should throw if the returned address of owner is zero yet the function does not implement logic to throw on such event.

Recommendation:

We advise to either implement the logic to throw when the returned owner's address is zero or remove the comments expecting this behaviour from the function to increase the legibility of the codebase.

Alleviation:

Alleviations were applied as of commit hash `6536d434b7c87c1cf0325917f811bf042e74ef50`.

AFT-10: Comparison with literal `false`

Type	Severity	Location
Gas Optimization	Informational	AavegotchiFacet.sol L455

Description:

The aforementioned line performs comparison with a boolean literal `false` which can be replaced with the negation of the expression to increase the legibility of the codebase.

Recommendation:

We advise to use the negation of expression inside the `require` statement instead of comparison with boolean.

```
require(!s.aavegotchiNamesUsed[_name], "AavegotchiFacet: Aavegotchi name used already");
```

Alleviation:

Alleviations were applied as of commit hash `6536d434b7c87c1cf0325917f811bf042e74ef50`.

AFT-11: Redundant Variable Initialization

Type	Severity	Location
Coding Style	Informational	AavegotchiFacet.sol L478

Description:

All variable types within Solidity are initialized to their default "empty" value, which is usually their zeroed out representation. Particularly:

- `uint` / `int` : All `uint` and `int` variable types are initialized at `0`
- `address` : All `address` types are initialized to `address(0)`
- `byte` : All `byte` types are initialized to their `byte(0)` representation
- `bool` : All `bool` types are initialized to `false`
- `ContractType` : All contract types (i.e. for a given `contract ERC20 {}` its contract type is `ERC20`) are initialized to their zeroed out address (i.e. for a given `contract ERC20 {}` its default value is `ERC20(address(0))`)
- `struct` : All `struct` types are initialized with all their members zeroed out according to this table

Recommendation:

We advise that the linked initialization statements are removed from the codebase to increase legibility.

Alleviation:

Alleviations were applied as of commit hash `6536d434b7c87c1cf0325917f811bf042e74ef50` .

AFT-12: `uint8` type can be changed to `uint256`

Type	Severity	Location
Gas Optimization	Informational	AavegotchiFacet.sol L479

Description:

The aforementioned line declares local variable of type `uint8` which is inefficient as local variables are not packed unlike storage variables. As EVM works with 32-byte values, the `uint8` type will be first unpacked to 32-byte and then be operated upon and hence it will be gas efficient to just use `uint256` instead of `uint8` on the aforementioned.

Recommendation:

We advise to the change type from `uint8` to `uint256` on the aforementioned line as it is gas efficient.

```
    for (uint256 index = 0; index < _values.length; index++) {...}
```

Alleviation:

Alleviations were applied as of commit hash `6536d434b7c87c1cf0325917f811bf042e74ef50` .

AFT-13: `require` statements can be substituted with `modifier`

Type	Severity	Location
Gas Optimization	Informational	AavegotchiFacet.sol L291, L501

Description:

The `require` statements on the aforementioned lines can be substituted with `modifier` to increase the legibility of the codebase.

Recommendation:

We advise to substitute the `require` statements with `modfier` on the aforementioned lines to increase the legibility of the codebase.

```
    modifier onlyStatusAavegotchi(uint256 _tokenId) {
       isStatusAavegotchi(_tokenId);
       _;
    }
```

```
    function isStatusAavegotchi(uint256 _tokenId) private {
       require(s.aavegotchis[_tokenId].status == LibAppStorage.STATUS_AAVEGOTCHI,
    "AavegotchiFacet: Must be claimed");
    }
```

The usage of modifier would be as followed.

```
    function func_name(uint256 _tokenId) onlyStatusAavegotchi(_tokenId) {...}
```

Alleviation:

The recommendation was not considered.

AFT-14: Documentation discrepancy

Type	Severity	Location
Logical Issue	Major	AavegotchiFacet.sol L581

Description:

The comment on `L27` of `LibAppStorage` suggests that the interaction count should be set to `0` when an Aavegotchi is transferred to a new owner yet the logic inside aforementioned function does not implement such behaviour.

Recommendation:

We advise to add the logic to set `interactionCount` of Aavegotchi to `0` in the aforementioned function.

```
    s.aavegotchis[_tokenId].interactionCount = 0;
```

Alleviation:

Alleviations were applied as of commit hash `6536d434b7c87c1cf0325917f811bf042e74ef50` by removing the comment.

CFT-01: Duplicate code

Type	Severity	Location
Coding Style	Informational	CollateralFacet.sol L20-L23

Description:

The struct definition on the aforementioned lines is also declared in contract `DAOFacet` on `L18-L21`. To observe the reusability of the code, this struct definition can be placed in the file `LibAppStorage` as it is imported in both of the contracts.

Recommendation:

We advise to put the struct definition in `LibAppStorage` to observe the resuability of code.

Alleviation:

Alleviations were applied as of commit hash `6536d434b7c87c1cf0325917f811bf042e74ef50` by putting struct in the LibAavegotchi.sol file.

CFT-02: Possibility of integer underflow

Type	Severity	Location
Arithmetic	Minor	CollateralFacet.sol L80

Description:

The aforementioned line performs check that the stake should be greater-than or equal-to `minimumStake` after it is decreased. If the expression `currentStake - _reduceAmount` underflow then the check still passes and the user is able to withdraw enough stake such that the actual stake would be less than the `minimumStake` .

Recommendation:

We recommend to either use `SafeMath` library from Openzeppelin or add a check asserting that the `_reduceAmount` is less than or equal to current stake.

```
    require(
       currentStake >= _reduceAmount,
       "currentStake cannot be less than the reduce amount"
    );
```

Alleviation:

Alleviations were applied by upgradting Solidity version to `0.8.1` which checks the arithmetic operations by default.

CFT-03: Incorrect word

Type	Severity	Location
Coding Style	Informational	CollateralFacet.sol L97

Description:

The comment on the aforementioned line incorrectly refers to `experience` as `essense`.

Recommendation:

We advise to rectify the comment and correctly mention the word `experience`.

```
//If the toId is different from the tokenId, then perform an experience transfer
```

Alleviation:

Alleviations were applied as of commit hash `6536d434b7c87c1cf0325917f811bf042e74ef50`.

DAO-01: `emit` keyword is missing before the event call

Type	Severity	Location
Language Specific	Minor	DAOFacet.sol L152

Description:

The event call on the aforementioned line is missing the `emit` keyword before it.

Recommendation:

We recommend to add the `emit` keyword before the event call.

```
    emit GameManagerTransferred(s.gameManager, _gameManager);
```

Alleviation:

Alleviations were applied as of commit hash `6536d434b7c87c1cf0325917f811bf042e74ef50` .

DAO-02: Possibility of integer overflow

Type	Severity	Location
Arithmetic	Minor	DAOFacet.sol L96-L97

Description:

The addition performed on `L96`, if overflown, can result in passing of the check on `L97` if the wrapped value is less than the `maxQuantity`.

Recommendation:

We advise to add a check asserting that the resultant `totalQuantity` of the addition operation is greater than `s.itemTypes[itemId].totalQuantity` or `SafeMath` library from Openzeppelin can be used.

```
require(
   totalQuantity >= s.itemTypes[itemId].totalQuantity,
   "totalQuantity has overflown"
);
```

The `SafeMath` library from Openzeppelin can be reffered to on the following link.

```
https://github.com/OpenZeppelin/openzeppelin-
contracts/blob/master/contracts/math/SafeMath.sol
```

Alleviation:

Alleviations were applied as of commit hash `6536d434b7c87c1cf0325917f811bf042e74ef50` by upgrading Solidity version to `0.8.1` which checks arithmetic operations by default.

SFT-01: Redundant storage read

Type	Severity	Location
Gas Optimization	Informational	ShopFacet.sol L57, L52

Description:

The storage read of `s.currentHauntId` on `L57` is redundant and inefficient as the same value is read from storage and stored in local variable on `L52`.

Recommendation:

We advise to read the `hauntId` from the local variable instead of storage on `L57` as reading from stack is significantly cheaper than reading from storage.

```
    uint16 hanutId = uint16(currentHauntId);
```

Alleviation:

Alleviations were applied as of commit hash `6536d434b7c87c1cf0325917f811bf042e74ef50`.

SFT-02: Possibility of integer overflow

Type	Severity	Location
Arithmetic	Minor	ShopFacet.sol L98

Description:

The addition operation on `L98`, if results in overflow of integer value then the `require` check on `L99` can possibly pass if the wrapped value is less than the `maxQuantity`.

Recommendation:

We advise to add a check asserting that the resultant value is greater than or equal to `itemType.totalQuantity` or `SafeMath` library from Openzeppelin can be used.

```
requrie(
   totalQuantity >= itemType.totalQuantity,
   "value has overflown"
);
```

The following link refers to `SafeMath` library.

```
https://github.com/OpenZeppelin/openzeppelin-
contracts/blob/master/contracts/math/SafeMath.sol
```

Alleviation:

Alleviations were applied as of commit hash `6536d434b7c87c1cf0325917f811bf042e74ef50` by upgrading Solidity version to `0.8.1` which checks the arithmetic operations by default.

SFT-03: Possibility of integer overflow

Type	Severity	Location
Arithmetic	Minor	ShopFacet.sol L123

Description:

The addition operation on `L123`, if results in overflow of integer value then the `require` check on `L124` can possibly pass if the wrapped value is less than the `maxQuantity`.

Recommendation:

We advise to add a check asserting that the resultant value is greater than or equal to `itemType.totalQuantity` or `SafeMath` library from Openzeppelin can be used.

```
requrie(
   totalQuantity >= itemType.totalQuantity,
   "value has overflown"
);
```

The following link refers to `SafeMath` library.

```
https://github.com/OpenZeppelin/openzeppelin-
contracts/blob/master/contracts/math/SafeMath.sol
```

Alleviation:

Alleviations were applied as of commit hash `6536d434b7c87c1cf0325917f811bf042e74ef50` by upgrading Solidity version to `0.8.1` which checks the arithmetic operations by default.

SFT-04: Inefficient Greater-Than Comparison w/ Zero

Type	Severity	Location
Gas Optimization	Informational	ShopFacet.sol L159

Description:

The linked greater-than comparisons with zero compare variables that are restrained to the non-negative integer range, meaning that the comparator can be changed to an inequality one which is more gas efficient.

Recommendation:

We advise that the above paradigm is applied to the linked greater-than statements.

Alleviation:

The relevant code part was remove rendering this exhibnit ineffectual.

VRF-01: Comparison with literal `false`

Type	Severity	Location
Gas Optimization	Informational	VRFFacet.sol L126

Description:

The aforementioned line performs comparison with boolean literal `false` which can be replaced with the negation of expression to increase the legibility of the codebase.

Recommendation:

We advise to replace literal comparison with `false` to the negation of expression on the aforementioned line.

```
    require(!vrf_ds.vrfPending, "VrfFacet: VRF call is pending");
```

Alleviation:

The relevant code part was removed rendering this exhibit ineffectual.

VRF-02: Comparison with literal `true`

Type	Severity	Location
Gas Optimization	Informational	VRFFacet.sol L153

Description:

The aforementioned line performs comparison with literal `true` which can be replaced with the expression itself to increase the legibility of the code.

Recommendation:

We advise to replace the comparison to literal `true` with the expression itself.

```
    require(vrf_ds.vrfPending, "VrfFacet: VRF is not pending");
```

Alleviation:

The relevant code part was changed rendering this exhibit ineffectual.

VRF-03: Returned `success` value is not checked of function call

Type	Severity	Location
Volatile Code	Minor	VRFFacet.sol L132

Description:

The aforementioned line performs function call whose returned `success` value is not checked.

Recommendation:

We advise to check the returned `success` value of the function call on the aforementioned line.

```
    require(
       im_link.transferAndCall(im_vrfCoordinator, vrf_ds.fee, abi.encode(vrf_ds.keyHash,
    0)),
       "call failed"
    )
```

Alleviation:

Alleviations were applied as of commit hash `6536d434b7c87c1cf0325917f811bf042e74ef50` .

VRF-04: `require` statement can be substituted with `modifier`

Type	Severity	Location
Gas Optimization	Informational	VRFFacet.sol L163, L171

Description:

The `require` statements on the aforementioned lines can be substituted with `modifier` to increase the legibility of the codebase.

Recommendation:

We advise to substitute the `require` statements on the aforementioned lines with `modifier`.

```
modifier onlyOwner() {
   isOwner();
   _;
}
```

```
function isOwner() private {
   require(msg.sender == LibDiamond.contractOwner(), "VrfFacet: Must be contract
owner");
}
```

The usage of `modifier` would be as followed.

```
function func_name() onlyOwner {...}
```

Alleviation:

Alleviations were applied as of commit hash `6536d434b7c87c1cf0325917f811bf042e74ef50`.

IFT-01: Incorrect code

Type	Severity	Location
Control Flow	Medium	ItemsFacet.sol L138

Description:

The aforementioned has incorrect condition in the `for` loop which results `bals` array always containing zero-values for all of its indices.

Recommendation:

We advise to rectify the condition part in `for` loop such that it correctly populate the `bals` array.

```
for (uint156 i; i < owners.length; i++) {...}
```

Alleviation:

Alleviations were applied as of commit hash `6536d434b7c87c1cf0325917f811bf042e74ef50`.

IFT-02: `uint256` can be used instead of `uint16`

Type	Severity	Location
Gas Optimization	Informational	ItemsFacet.sol L147

Description:

The aforementioned line declares local variable of type `uint16` which is inefficient as local variables are not packed unlike storage variables. As EVM works with 32-byte values, the `uint16` type will be first unpacked to 32-byte and then be operated upon and hence it will be gas efficient to just use `uint256` instead of `uint16` on the aforementioned.

Recommendation:

We recommend to change the type from `uint16` to `uint256` on the aforementioned line as it will be gas efficient.

```
    for (uint256 i; i < 16; i++) {...}
```

Alleviation:

Alleviations were applied as of commit hash `6536d434b7c87c1cf0325917f811bf042e74ef50`.

IFT-03: Inefficient function implementation

Type	Severity	Location
Gas Optimization	Informational	ItemsFacet.sol L181

Description:

The function on the aforementioned line manually copies all the `itemTypes` to a memory array and then return it. The function will behave the same if the `itemType` are directly returned from the storage instead of manually copying them.

Recommendation:

We advise to directly return the `itemTypes` from the body of the function as it will be gas efficient.

```
function getItemTypes() external view returns (ItemType[] memory itemTypes_) {
   return s.itemTypes;
}
```

Alleviation:

The relevant code is removed rendering this exhibit ineffectual.

IFT-04: Comparison with a literal `true`

Type	Severity	Location
Gas Optimization	Informational	ItemsFacet.sol L404

Description:

The aforementioned line performs comparison with a literal `true` which can be replaced with the expression itself to increase the legibility of the codebase.

Recommendation:

We advise to replace the comparison with boolean literal `true` with the expression itself on the aforementioned line.

```
require(canBeEquipped, "ItemsFacet: Wearable cannot be equipped in this collateral type");
```

Alleviation:

Alleviations were applied as of commit hash `6536d434b7c87c1cf0325917f811bf042e74ef50`.

IFT-05: Duplicate code can be extracted to a function

Type	Severity	Location
Gas Optimization	Informational	ItemsFacet.sol L296-L311, L343-L358

Description:

The code blocks on the aforementioned lines are a duplicate and can be extracted to a single function to increase the legibility of the codebase and as well as reduce the bytecode footprint of the contract to save gas cost associated with contract deployment.

Recommendation:

We advise to extract the duplicate code on the aforementioned lines to a function to observe reusability of the code. The extracted function can be called in place of the aforementioned code.

```
function isValidSender(
   address _fromContract,
   uint256 _fromTokenId
) private {
   if (_fromContract == address(this)) {
            address owner = s.aavegotchis[_fromTokenId].owner;
            require(
                msg.sender == owner || s.operators[owner][msg.sender] || msg.sender ==
s.approved[_fromTokenId],
                "Items: Not owner and not approved to transfer"
            );
            require(s.aavegotchis[_fromTokenId].unlockTime <= block.timestamp, "Items:
Only callable on unlocked Aavegotchis");
        } else {
            address owner = IERC721(_fromContract).ownerOf(_fromTokenId);
            require(
                owner == msg.sender ||
                    IERC721(_fromContract).getApproved(_fromTokenId) == msg.sender ||
                    IERC721(_fromContract).isApprovedForAll(owner, msg.sender),
                "Items: Not owner and not approved to transfer"
            );
        }
}
```

Alleviation:

Alleviations were applied as of commit hash `6536d434b7c87c1cf0325917f811bf042e74ef50`.

IFT-06: Duplicate code can be extracted to a function

Type	Severity	Location
Gas Optimization	Informational	ItemsFacet.sol L315-L320, L362-L367

Description:

The code on the aforementioned lines is duplicate and can be extracted to a function to increase the legibility of the codebase and as well as reduce bytecode footrpint of the contract resulting in reduced gas cost associated with deployment.

Recommendation:

We recommend to extract the duplicate code into a separate function to observe the reusability of the codebase.

```
    function checkEquippedWearables(
       address _fromContract,
       uint256 _fromTokenId,
       uint256 _id
    ) private {
       if (bal == 0 && _fromContract == address(this)) {
                uint256 l_equippedWearables =
    s.aavegotchis[_fromTokenId].equippedWearables;
                for (uint256 i; i < 16; i++) {
                    require(uint16(l_equippedWearables >> (i * 16)) != _id, "Items: Cannot
    transfer wearable that is equipped");
                }
            }
    }
```

Alleviation:

Alleviations were applied as of commit hash `6536d434b7c87c1cf0325917f811bf042e74ef50` .

ADD-01: Ether locking in AavegotchiDiamond

Type	Severity	Location
Volatile Code	Medium	AavegotchiDiamond.sol L88

Description:

The `receive` function on the aforementioned line allows the diamond proxy contract to receive plain `ether` yet none of the facet contract has any functionality to deal with the received ethers. Any sent ethers would be locked in the contract unless an implementation contract is introduced for the withdrawal of the ethers.

Recommendation:

We advise to remove the declaration of the `receive` function so the diamond proxy contract does not allow the receiving of plain ether.

Alleviation:

Alleviations were applied as of commit hash `6536d434b7c87c1cf0325917f811bf042e74ef50` .

LDD-01: Inefficient storage struct layout

Type	Severity	Location
Gas Optimization	Informational	LibDiamond.sol L27, L33

Description:

The properties of the struct on the aforementioned lines can placed next to each other to tight pack them in a single 32-byte storage slot. Currently, both of the properties are occupying 32-byte slots each.

Recommendation:

We advise to place the aforementioned struct proerties next to each other to tight pack them in a single 32-byte storage slot to save gas cost associated with the additional storage slot.

```
struct DiamondStrorage {
    ...
    // The number of function selectors in selectorSlots
         uint16 selectorCount;
    // owner of the contract
    address contractOwner;
}
```

Alleviation:

The relevant code is removed rendering this exhibit ineffectual.

LDD-02: Redundant `require` statement

Type	Severity	Location
Gas Optimization	Informational	LibDiamond.sol L61

Description:

The `require` statement on the aforementioned line can be replace with the call to `enforceIsContractOwner` to increase the legibility of the codebase. It will also be gas efficient as it will decrease bytecode footprint of the contract and any function making use of the modifier.

Recommendation:

We advise to make use of the call to function `enforceIsContractOwner` instead of the `require` statement.

```
modifier onlyOwner {
   enforceIsContractOwner();
        _;
}
```

Alleviation:

Alleviations were applied as of commit hash `6536d434b7c87c1cf0325917f811bf042e74ef50`.

LDD-03: Inefficient Greater-Than Comparison w/ Zero

Type	Severity	Location
Gas Optimization	Informational	LibDiamond.sol L240

Description:

The linked greater-than comparisons with zero compare variables that are restrained to the non-negative integer range, meaning that the comparator can be changed to an inequality one which is more gas efficient.

Recommendation:

We advise that the above paradigm is applied to the linked greater-than statements.

Alleviation:

Alleviations were applied as of commit hash `6536d434b7c87c1cf0325917f811bf042e74ef50`.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that 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.

Arithmetic

Arithmetic exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

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

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an in-storage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete`.

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.