CERTIK

Security Assessment

Hashstack

May 2nd, 2022

Table of Contents

Summary

This report has been prepared for Hashstack to discover issues and vulnerabilities in the source code of the Hashstack 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.

Please note that there are two repositories in development:

1. https://github.com/0xHashstack/open_contracts/ This repository is used for initial auditing. The commit hashes in finding locations and initial client responses are from this repository.
2. https://github.com/0xHashstack/Open-contracts/ This repository is used for client response. The commit hashes in secondary client responses are from this repository. These commit hashes are usually covered by angel brackets "<>"

Overview

Project Summary

Project Name	Hashstack
Platform	Other
Language	Solidity
Codebase	https://github.com/0xHashstack/Open-contracts
Commit	f1a425dbe0c1e431da6563e89c675d3fb95bc2e1

Audit Summary

Delivery Date	May 02, 2022 UTC
Audit Methodology	Static Analysis, Manual Review

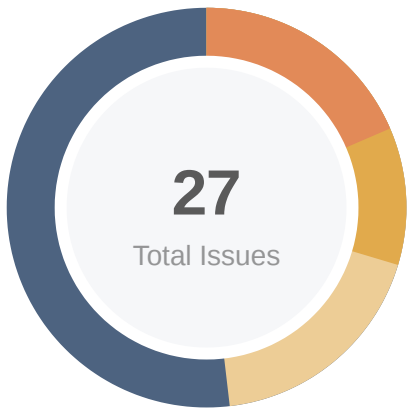
Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Mitigated	Partially Resolved	Resolved
Critical	0	0	0	0	0	0	0
Major	5	0	0	2	1	0	2
Medium	3	0	0	0	0	0	3
Minor	5	0	0	2	0	0	3
Informational	14	0	0	4	0	1	9
Discussion	0	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
DLF	facets/DiamondLoupeFacet.sol	d5e8d810d3113da805ffab25943ce21a71e8f16f5b5aa8cb00671e421c830ac8
LDH	libraries/LibDiamond.sol	1c1be7ba988ff7d4a0317085694e82d8cae02eebb46460d6f12bfbd730f20a46
ARH	AccessRegistry.sol	630895c42f42649d81572d57208a788f58c56cd16d7164fa60f66dc47521b4ed
LHK	Loan1.sol	6c25e279d344d81cf01b1396199341b75cce0bb5355221b46bc3db7951cb9a53
DCF	facets/DiamondCutFacet.sol	0cb645177d4807c1a2bf5de1876885e98441650214b312f8b76df3562114591f
DHC	Deposit.sol	a73be931272eecc35484afee379f17c4e0c40c14eff1e27c62d696ec5a80b9c6
HCK	facets	
LHC	Loan.sol	f320e094387b712355dee9e4ef0ad2b13ee57ad6ce70211b3902d1069b242d64
DIH	facets/DiamondInit.sol	cccb80b05b688fbf1d85917a8d2c8aced38f14a4cc3d2bb1e0e88fc4f476e99f
HCP	libraries	

Findings



Critical	**0** (0.00%)
Major	**5** (18.52%)
Medium	**3** (11.11%)
Minor	**5** (18.52%)
Informational	**14** (51.85%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
GLOBAL-01	Centralization Related Risks	**Centralization / Privilege**	**Major**	Acknowledged
DHC-01	Inconsistency On `nonReentrant` Modifiers	Volatile Code	Minor	Resolved
DIH-01	The Initialize Function Can Be Called Multiple Times	Logical Issue	Informational	Acknowledged
HCK-01	Invalid Caller Checks In `authXXX` Modifiers	Logical Issue	Major	Resolved
HCK-02	Negligence Of Return Value Of `_hasDeposit()`	Control Flow	Medium	Resolved
HCK-03	Third Party Dependencies	Volatile Code	Minor	Acknowledged
HCK-04	Typos In Codes And Comments	Coding Style	Informational	Partially Resolved
HCK-05	Redundant `require` Statements	Gas Optimization	Informational	Resolved
LDH-01	Risky `approveFrom()` Function For BEP20/EIP20	Volatile Code	Major	Resolved
LDH-02	Unrestricted Privileged Function `_transferAnyBEP20()`	Control Flow	Major	Acknowledged
LDH-03	Transferring To `contractOwner` Instead Of Reserve Contract	**Centralization / Privilege**	**Major**	Mitigated

ID	Title	Category	Severity	Status
LDH-04	Returning An Empty Value In Function `_swap()`	Logical Issue	Medium	Resolved
LDH-05	Meaningless `amountOutMin` Parameter For `PancakeRouter01.swapExactTokensForTokens()`	Volatile Code	Minor	Acknowledged
LDH-06	Wrong BEP20/EIP20 Application	Volatile Code	Minor	Resolved
LDH-07	Uninitialized State Variables `bytes32 adminXXX`	Logical Issue	Minor	Resolved
LDH-08	Testnet Addresses	Volatile Code	Informational	Acknowledged
LDH-09	Redundant Data Structure	Gas Optimization	Informational	Resolved
LDH-10	If-Else Statement Logics	Logical Issue	Informational	Resolved
LDH-11	Missing Validations	Logical Issue	Informational	Resolved
LDH-12	Uninitialized Local Variable	Volatile Code	Informational	Resolved
LDH-13	Division Before Multiplication	Mathematical Operations	Informational	Resolved
LDH-14	Questions On Market Support	Data Flow	Informational	Resolved
LDH-15	Unnecessary Function Parameter	Gas Optimization	Informational	Resolved
LDH-16	_permissibleCDR() Logic	Logical Issue	Informational	Acknowledged
LDH-17	Undefined Fee Structure	Inconsistency	Informational	Acknowledged
LDH-18	State Variable `priceData` Is Not Regularly Updated	Volatile Code	Informational	Resolved
LHK-01	Calling Functions `liquidation()` And `permissibleWithdrawal()` In The Same Block	Volatile Code	Medium	Resolved

GLOBAL-01 | Centralization Related Risks

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

Description

The following contracts used modifiers for the listed functions that would allow certain accounts to perform highly privileged actions:

Deposit.sol: authDeposit

- `pauseDeposit`
- `unpauseDeposit`

Loan.sol: authLoan

- `pauseLoan`
- `unpauseLoan`

Loan1.sol: authLoan1

- `liquidation`
- `pauseLoan1`
- `unpauseLoan1`

AccessRegistry.sol: onlyAdmin

- `addRole`
- `removeRole`
- `addAdminRole`
- `removeAdminRole`
- `adminRoleTransfer`
- `adminRoleRenounce`
- `pauseAccessRegistry`
- `unpauseAccessRegistry`

LibDiamond.sol: authContract(TOKENLIST_ID)

- `_addMarketSupport`

- `_removeMarketSupport`
- `_updateMarketSupport`
- `_addMarket2Support`
- `_removeMarket2Support`
- `_updateMarket2Support`

LibDiamond.sol: authContract(COMPTROLLER_ID)

- `_updateApy`
- `_setCommitment`
- `_updateApr`

LibDiamond.sol: authContract(DEPOSIT_ID)

- `_withdrawDeposit`
- `_accruedYield`
- `_processNewDeposit`
- `_processDeposit`
- `_accountBalance`
- `_updateSavingsBalance`
- `_updateReservesDeposit`
- `_createNewDeposit`
- `_addToDeposit`
- `_convertYield`

LibDiamond.sol: authContract(LOAN_ID)

- `_swapToLoan`
- `_withdrawCollateral`
- `_swapLoan`
- `_repayLoan`

LibDiamond.sol: authContract(LOAN1_ID)

- `_addCollateral`
- `_loanRequest`
- `_permissibleWithdrawal`
- `_liquidation`

LibDiamond.sol: authContract(RESERVE_ID)

- `_collateralTransfer`
- `_transferAnyBEP20`

LibDiamond.sol: authContract(ACCESSREGISTRY_ID)

- `_addRole`
- `_revokeRole`
- `_addAdminRole`
- `_revokeAdmin`

Any compromise to the privileged account may allow a hacker to take advantage of this authority and perform highly privileged actions.

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 (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.

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.

Alleviation

[Hashstack Team]:

We still intend to keep those functions as they are required for hashstack's business model.

The reason we need these types of functions is to protect our users in case of any hacks or exploits happening in near future. Where in we can simply pause the exploited contract such that we can protect user funds from malicious actors.

DHC-01 | Inconsistency On `nonReentrant` Modifiers

Category	Severity	Location	Status
Volatile Code	Minor	Deposit.sol: 33	Resolved

Description

In Deposit, the `nonReentrant` modifier is not added on function `savingsBalance()`;

In Loan, the `nonReentrant` modifier is not added on functions `withdrawCollateral()` and `repayLoans()`;

In Loan1, the `nonReentrant` modifier is not added on functions `addCollateral()` and `permissibleWithdraw()`;

In AccessRegistry, the `nonReentrant` modifier is not added on functions `addRole()`, `removeRole()`, `addAdminRole()`, `removeAdminRole()` , `adminRoleTransfer()` and `adminRoleRenounce`.

Recommendation

Recommend keeping consistency on the `nonReentrant` modifiers on external/public functions.

Alleviation

Some of the highlighted instances were deleted (`savingsBalance()`) or fixed in commit `f0d00d4095e77b7e69efe0b920ffae5077304630`. However, the functions in AccessRegistry were still missing the `nonReentrant` modifier.

[Hashstack Team]:

Majority of functions already had non reentrant. I suggest certik to review the staging branch https://github.com/0xHashstack/Open-contracts/tree/staging
<a6bd967669a2062ce8b69cf350c05b6a61a20725>

DIH-01 | The Initialize Function Can Be Called Multiple Times

Category	Severity	Location	Status
Logical Issue	Informational	facets/DiamondInit.sol: 17	Acknowledged

Description

The `init()` function can be called multiple times.

Although no impact could be determined at the current stage since the `init()` function only set the `supportInterface` fields, it might be a problem if some other state variables are added according to the comments.

Recommendation

It is recommended to make sure that initialize functions can only be called once.

Alleviation

[Hashstack Team]:

Diamond init getting called through diamond cut faucet.

The init() function resides inside the diamond init contract which can only be called through diamond-cut while upgrading. This upgrading process can only be done by an authorized address which in our case would be a mutlisig wallet.

Reference - https://github.com/0xHashstack/Open-contracts/blob/staging/scripts/deploy_all.js#L151

HCK-01 | Invalid Caller Checks In `authXXX` Modifiers

Category	Severity	Location	Status
Logical Issue	Major	AccessRegistry.sol: 108; Loan1.sol: 77; Loan.sol: 81; Deposit.sol: 121; libraries/LibDiamond.sol	Resolved

Description

The `authDeposit()`, `authLoan()`, `authLoan1()`, `onlyAdmin` and `authContract()` modifiers were found to be missing checks regarding the current transaction. Therefore, they were not actually performing any valid authorization checks; any account would be able to call any of the supposedly protected functions.

Recommendation

Implement a proper authorization mechanism.

Alleviation

This issue was found to be fixed in commit `f0d00d4095e77b7e69efe0b920ffae5077304630`.

HCK-02 | Negligence Of Return Value Of `_hasDeposit()`

Category	Severity	Location	Status
Control Flow	Medium	Deposit.sol: 69~70; libraries/LibDiamond.sol: 847	Resolved

Description

In LibDiamond, function `_accruedYield()` calls `_hasDeposit()` and ignores its return value; in Deposit, function `hasDeposit()` calls `_hasDeposit()`, ignores its return value and always returns true.

Note that unlike the other `_hasXXX()` functions, `_hasDeposit()` does not have any require checks inside, the functionality fully relies on the return value.

```
function _hasDeposit( address _account, bytes32 _market, bytes32 _commitment) internal
view returns (bool ret) {
    DiamondStorage storage ds = diamondStorage();
    ret = ds.indDepositRecord[_account][_market][_commitment].id != 0;
}
```

Recommendation

Recommend properly handling the return value of `_hasDeposit()` in functions that call it, or use require statements instead of return values to meet the design specifications.

Alleviation

The `hasDeposit()` function is still ignoring the returned boolean by `_hasDeposit()` in commit `f0d00d4095e77b7e69efe0b920ffae5077304630`.

[Hashstack Team]:

Issue acknowledged. Changes have been reflected in the commit hash <4b90bd6887644c3135e7114f0486df7a61bbfd95>.

HCK-03 | Third Party Dependencies

Category	Severity	Location	Status
Volatile Code	Minor	libraries/LibDiamond.sol; facets	Acknowledged

Description

The contract is serving as the underlying entity to interact with the third party protocol `PancakeSwap` and `Chainlink`. The scope of the audit treats 3rd party entities as black boxes and assume their functional correctness. However, in the real world, 3rd parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of 3rd parties can possibly create severe impacts, such as increasing fees of 3rd parties, migrating to new LP pools, etc.

Furthermore, we noticed that the on-chain data storage is based on the Diamonds Multi-Facet Proxy EIP-2535. Currently the EIP status of EIP-2535 is still “Last Call”.

For more information visit: https://eips.ethereum.org/EIPS/eip-2535.

Recommendation

We understand that the business logic of 0xHaskstack requires the usage of Chainlink, PancakeSwap and implementation of EIP-2535. We encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed. Additionally, we would like to recommend the client to monitor the EIP status and making responsive code updates if necessary.

Alleviation

[Hashstack Team]:

Issue acknowledged. I won't make any changes for the current version.

We are constantly monitoring the statuses of 3rd parties to mitigate any side effects when unexpected activities are observed. Along with that we are constantly monitoring EIP status as well

HCK-04 | Typos In Codes And Comments

Category	Severity	Location	Status
Coding Style	Informational	libraries/LibDiamond.sol: 83, 84, 96, 104, 135, 300, 301, 302, 673, 678, 696, 1384, 1407, 1470, 1624, 1788; Deposit.sol: 114; facets/DiamondInit.sol: 9, 11	Partially Resolved

Description

There are several typos in the code and comments.

Recommendation

We recommend correcting all typos in the contract.

Alleviation

Some typos are still present.

[Hashstack Team]:

Issue acknowledged. Changes have been reflected in the commit hash <ab05776e2591636ace02e849fbaa35c1d17427e8>.

HCK-05 | Redundant `require` Statements

Category	Severity	Location	Status
Gas Optimization	Informational	libraries/LibDiamond.sol: 1625, 1633, 1976; facets/DiamondLoupeFacet.sol: 27	Resolved

Description

The linked `require` statements are redundant.

For example, the `numFacetSelectors` of `DiamondLoupeFacet.sol` in line 27 is typeof `uint8[]`, so the elements' range is [0, 255]. In solidity 0.8.0+ integer overflows would revert by default, therefore the check would be redundant.

Regarding the `require` statement on line 1625 of the file `LibDiamond.sol`, it is considered redundant because it is checked again on line 1633.

```
// probably will never have more than 256 functions from one facet contract
require(numFacetSelectors[facetIndex] < 255);
numFacetSelectors[facetIndex]++;
```

Recommendation

We advise the team to remove the redundant require statements.

Alleviation

Fixed in commit hash `f0d00d4095e77b7e69efe0b920ffae5077304630`.

LDH-01 | Risky `approveFrom()` Function For BEP20/EIP20

Category	Severity	Location	Status
Volatile Code	Major	libraries/LibDiamond.sol: 737, 837, 1047, 1654, 1739, 1846, 1966, 1971~1972	Resolved

Description

`approveFrom()` is not a function declared in BEP20. We understand the function aims to enable the caller/msg.sender to add an `amount` of allowance from an `owner` address for a `spender` address. If the `approveFrom()` function can be freely called, anyone can 1) call it and change allowance, 2) call `transferFrom()` and transfer money at their will. Reference: https://github.com/binance-chain/BEPs/blob/master/BEP20.md

Recommendation

Recommend disabling this feature or adding proper access control on the `approveFrom()` function. Please also see Global-01 “Centralization Related Risks” for recommendations on privileged functions.

Alleviation

Even though the project is not making use of the `approveFrom()` function, its usage has only been commented out and the interface `IBEP20` still has it in its code.

[Hashstack Team]:

Issue acknowledged. Changes have been reflected in the commit hash <74f853a421374c370b0cee263d77127da5759529>.

LDH-02 | Unrestricted Privileged Function `_transferAnyBEP20()`

Category	Severity	Location	Status
Control Flow	Major	libraries/LibDiamond.sol: 1970~1971	Acknowledged

Description

Function `_transferAnyBEP20()` is supposed to be a privileged function that needs to be restricted by admin only permissions. However, the visibility is `internal`, and the only check of it is `authContract()`. The `internal` visibility enables any in-contract/inherited functions to call it. The `authContract()` modifier does not have restrictions on the function caller. This finding is related to two other findings:

1. HCK-01: Invalid Caller Checks in Modifiers
2. LDH-01: Risky `approveFrom()` Function for BEP20/EIP20

Recommendation

Recommend adding proper permission restrictions, review and fix the two related findings. Please also see GLOBAL-01 to see recommendations on Centralization Related Risks.

Alleviation

[Hashstack Team]:

Well in case a user sends funds to one of our contracts by mistake or sends wrong BEP20 from what he/she desired, we can use this function to recover their funds from our contracts and send the back.

Also, in case of any hack if we wish to return funds back to users we can do so using this function.

LDH-03 | Transferring To `contractOwner` Instead Of Reserve Contract

Category	Severity	Location	Status
Centralization / Privilege	**Major**	libraries/LibDiamond.sol: 838~839, 1048, 1655, 1740, 1810, 1847, 1847, 1967	Mitigated

Description

In functions `_withdrawDeposit()`, `_createNewDeposit()`, `_addCollateral()`, `_loanRequest()`, `_repayLoan()` and `_collateralTransfer()`, all assets are either transferred to or transferred from the `contractOwner` address. Our understanding is that the `contractOwner` address is an EOA, so it seems the address to manage and receive the assets should be the reserve contract address by design.

Also, the getter function `contractOwner()` in LibDiamond is never used.

Recommendation

Recommend switching the contract owner EOA to the reserve contract address with proper decentralization efforts (see GLOBAL-01).

Alleviation

The mentioned transfer functions are not sending funds to the contract owner anymore, but to the contracts.

[Hashstack Team]:

Currently, for all the functions, funds are getting transferred to our contract(reserve). And the getter function contractOwner() has been removed.

LDH-04 | Returning An Empty Value In Function `_swap()`

Category	Severity	Location	Status
Logical Issue	Medium	libraries/LibDiamond.sol: 760	Resolved

Description

`receivedAmount`, the return value of function `_swap()`, is never assigned. It seems the original design is to use the commented out ParaSwap, but now seems the design changed to use PancakeSwap, `receivedAmount` should be assigned by the return value of `swapExactTokensForTokens()`?

The impact is relatively high, since the return value of `_swap()` is widely used in functions `_swapToLoanProcess()`, `_repaymentProcess()`, `_swapLoan()`, `_repayLoan()` and `_liquidation()`.

Recommendation

Review the `_swap()` function to make sure it returns the appropriate value.

Alleviation

The `_swap()` function is returning the swapped amount as indicated by the team and observed in commit `f0d00d4095e77b7e69efe0b920ffae5077304630`.

LDH-05 | Meaningless `amountOutMin` Parameter For `PancakeRouter01.swapExactTokensForTokens()`

Category	Severity	Location	Status
Volatile Code	Minor	libraries/LibDiamond.sol: 755~756	Acknowledged

Description

In function `_swap()`, the call of `PancakeRouter.swapExactTokensForTokens()` passes the return value of `_getAmountOutMin()` to be the parameter of `amountOutMin`. Function `_getAmountOutMin()` calls `PancakeRouter01.getAmountsOut()` to retrieve the `amountOutMin` value, and `PancakeRouter01.getAmountsOut()` calls `PancakeLibrary.getAmountsOut()`. However, in `swapExactTokensForTokens()`, the input parameter `amountOutMin` is used to compared with the return value of `PancakeLibrary.getAmountsOut()`. That is to say the return value of `PancakeLibrary.getAmountsOut()` is compared with itself to check whether the `amountOutMin` is valid, which is meaningless.

Recommendation

Recommend tuning a value or a function to be used as the `amountOutMin` threshold, properly testing the threshold to make sure that it meets the design specifications, and documenting the threshold to reach the community consensus. Reference: PancakeRouter01.sol

Alleviation

[Hashstack Team]:

amountOutMin is used in swap function thereby we need it.

[CertiK]:

Agreed, `amountOutMin` is a parameter for the swap function call. However, this finding is trying to argue that the value of `amountOutMin` passing to the swap function should be a well-tuned customized value such that the `amountOutMin` threshold make sense to the design specifications.

[Hashstack Team]:

We are going to show amountOutMin on the web app such that it gives transparency and trust to our users. Apart from that we already are using amountOutMin in various function and have some future plans with it

so we intend to keep it.

LDH-06 | Wrong BEP20/EIP20 Application

Category	Severity	Location	Status
Volatile Code	Minor	libraries/LibDiamond.sol	Resolved

Description

According to EIP-20, functions `transfer()`, `transferFrom()`, and `approve()` should always have a `bool` return value, for the ERC20 caller to handle, as the callers must not assume that `false` is never returned.

The following functions of the `LibDiamond.sol` contract were found to be affected:

- `_swap()`:
  - `transferFrom()`
  - `approve()`
- `_withdrawDeposit()`:
  - `transferFrom()`
- `_createNewDeposit()`:
  - `transferFrom()`
- `_addCollateral()`:
  - `transferFrom()`
- `_loanRequest()`:
  - `transferFrom()`
- `_repayLoan()`:
  - `transferFrom()`
- `_permissibleWithdrawal()`:
  - `transfer()`
- `_collateralTransfer()`:
  - `transferFrom()`
- `_transferAnyBEP20()`:
  - `transferFrom()`

Recommendation

Make sure the returned values of the EIP-20 functions `transfer()`, `transferFrom()`, and `approve()` are properly handled.

Alleviation

In commit `f0d00d4095e77b7e69efe0b920ffae5077304630`, all EIP-20 functions `transfer()`, `transferFrom()`, and `approve()` are returning a boolean as indicated by the Hashstack team.

LDH-07 | Uninitialized State Variables `bytes32 adminXXX`

Category	Severity	Location	Status
Logical Issue	Minor	libraries/LibDiamond.sol: 191~193, 205~207, 223~229, 239~241, 246~250, 269~271	Resolved

Description

The state variables, `adminTokenList`, `adminComptroller`, `adminLiquidator`, `adminOpenOracle`, `adminLoan`, `adminLoan1`, `adminReserve`, are used in the second condition of the modifiers `authXXX` in the wrapper contracts.

However, these `bytes32` type state variables as well as the corresponding `address` type state variables, are never initialized, which means the modifiers `authXXX` in the wrapper contracts are now only checking whether `LibDiamond._hasAdminRole(ds.superAdmin, ds.contractOwner)` is `true`.

Recommendation

Recommend properly initializing the state variables and making sure the implementation meets the specifications.

Alleviation

The Hashstack team indicated the following regarding the uninitialized state variables: "Diamond init is getting called through diamond cut and state variables are initialized now".

LDH-08 | Testnet Addresses

Category	Severity	Location	Status
Volatile Code	Informational	libraries/LibDiamond.sol: 36~37	Acknowledged

Description

Constant state variables `PANCAKESWAP_ROUTER_ADDRESS` and `WBNB` are using the testnet addresses

Recommendation

Recommend double check and update to mainnet addresses when deployment.

Alleviation

[Hashstack Team]:

We will update the addresses before pushing them to the mainnet.

LDH-09 | Redundant Data Structure

Category	Severity	Location	Status
Gas Optimization	Informational	libraries/LibDiamond.sol: 166, 171	Resolved

Description

`_role` field in struct `RoleData` and `_adminRole` field in struct `AdminRoleData` are never used. The AccessRegistry state variables in library LibDiamond `_roles` and `_adminRoles` are mapping `bytes32` to `RoleData` and `AdminRoleData` respectively. Thus it seems the `byte32` type fields in the two AccessRegistry structs are never used.

```
// ========== AccessRegistry structs =============
    struct RoleData {
        mapping(address => bool) _members;
        bytes32 _role;
    }

    struct AdminRoleData {
        mapping(address => bool) _adminMembers;
        bytes32 _adminRole;
    }
```

Recommendation

We advise the client to remove the aforementioned code if there is no further plan to use this data type.

Alleviation

The two structures were still present in commit `f0d00d4095e77b7e69efe0b920ffae5077304630`.

[Hashstack Team]:

The two structs are needed in near future and some of the variables in it are still used, although we did eliminate the unused variables <72e28f6bc106190f102b9b199f94b463bcf5391c>

LDH-10 | If-Else Statement Logics

Category	Severity	Location	Status
Logical Issue	Informational	libraries/LibDiamond.sol: 890~896, 905~925, 1052~1058, 1066~1068, 1072	Resolved

Description

In the function `_processNewDeposit()`, the two “if-else” blocks could be simplified and some of the assignments could even go outside the conditional logic:

The first instance of complex logics was found in the next block:

```
890     uint id;
891
892     if (savingsAccount.deposits.length == 0) {
893         id = 1;
894     } else {
895         id = savingsAccount.deposits.length + 1;
896     }
```

And, the second instance was the following:

```
905   if (_commitment != _getCommitment(0)) {
906       yield.id = id;
907       yield.market = bytes32(_market);
908       yield.oldLengthAccruedYield = _getApyTimeLength(_commitment);
909       yield.oldTime = block.timestamp;
910       yield.accruedYield = 0;
911       yield.isTimelockApplicable = true;
912       yield.isTimelockActivated=  false;
913       yield.timelockValidity = 86400;
914       yield.activationTime = 0;
915   } else if (_commitment == _getCommitment(0)) {
916       yield.id=  id;
917       yield.market=_market;
918       yield.oldLengthAccruedYield = _getApyTimeLength(_commitment);
919       yield.oldTime = block.timestamp;
920       yield.accruedYield = 0;
921       yield.isTimelockApplicable = false;
922       yield.isTimelockActivated=  true;
923       yield.timelockValidity = 0;
924       yield.activationTime = 0;
925   }
```

Recommendation

A simpler alternative for both complex `if-else` instances would be:

```
    uint id = savingsAccount.deposits.length + 1;
```

and:

```
      yield.id = id;
      Yield.market = _market;
      yield.oldLengthAccruedYield = _getApyTimeLength(_commitment);
      yield.oldTime = block.timestamp;
      yield.accruedYield = 0;
      yield.activationTime = 0;
   if (_commitment != _getCommitment(0)) {
      yield.isTimelockApplicable = true;
      yield.isTimelockActivated =  false;
      yield.timelockValidity = 86400;
   } else {
      yield.isTimelockApplicable = false;
      yield.isTimelockActivated =  true;
      yield.timelockValidity = 0;
   }
```

Alleviation

[Hashstack Team]:

Issue acknowledged. Changes have been reflected in the commit hash <a00e84b44c9387c547a1e180a4b608983dec3642>

LDH-11 | Missing Validations

Category	Severity	Location	Status
Logical Issue	Informational	libraries/LibDiamond.sol: 392, 407, 427	Resolved

Description

The `_addMarketSupport()`, `_removeMarketSupport` and `_updateMarketSupport()` functions in LibDiamond.sol are not checking whether the markets are already supported or not.

Recommendation

Consider incorporating a verification such as `_isMarketSupported()` before adding, removing and updating the market support.

Alleviation

As of commit `f0d00d4095e77b7e69efe0b920ffae5077304630`, the validations are still missing.

[Hashstack Team]:

The validation is being done but rather than calling _isMarketSupported we are calling the base logic inside that function in the form of ds.tokenSupportCheck[_market] = false;

reference commit hash: <81ae4a6c98f05f19549dab7852c1b1b141808ce8>

We are now calling _isMarketSupported not only for primary markets but also for secondary markets.

LDH-12 | Uninitialized Local Variable

Category	Severity	Location	Status
Volatile Code	Informational	libraries/LibDiamond.sol	Resolved

Description

There are several local variables that are possible not to be initialized, and thus the variables are possible to be used with empty values. Here is the list of the uninitialized variables: LibDiamond:

- `_swap()`:
  - `addrFromMarket`
  - `addrToMarket`
- `_accountBalance()`:
  - `_savingsBalance`
- `_collateralTransfer()`:
  - `collateralMarket`
- `_repayLoan()`:
  - `_swappedAmount`
- `_accruedYield()`:
  - `aggregateYield`
- `_collateralTransfer()`:
  - `collateralAmount`
- `convertYield()`:
  - `_amount`
- `swapToLoan()`:
  - `_swappedAmount`

Recommendation

Recommend initializing the local variable to an acceptable default value

LDH-13 | Division Before Multiplication

Category	Severity	Location	Status
Mathematical Operations	Informational	libraries/LibDiamond.sol: 639, 643, 649, 679, 683, 689	Resolved

Description

Mathematical operations in functions `_calcAPR()` and `_calcAPY()` perform divisions before multiplications. Performing multiplication before division can sometimes avoid loss of precision.

Recommendation

Recommend applying multiplications before divisions to avoid potential precision loss, and adding proper tests for the functions with massive math calculations.

LDH-14 | Questions On Market Support

Category	Severity	Location	Status
Data Flow	Informational	libraries/LibDiamond.sol: 392, 407, 427	Resolved

Description

In LibDiamond, there are three functions managing the state variables of the markets, `_addMarketSupport()`, `_removeMarketSupport()` and `_updateMarketSupport()`. We have two questions regarding the functionalities:

1. We noticed that `marketData.minAmount` is only updated in `_addMarketSupport()`, but not in `_updateMarketSupport()`, is that an intended behavior?
2. After a market is added, under what circumstances the already added market **can** be removed or updated? And under what circumstances the already added market **cannot** be removed or updated?

Recommendation

Recommend properly documenting the control flow and use cases of the market support and adding `require` statements to make sure the code logics meet the design specifications. It would be risky if the control flows of the sensitive state variables are not clear.

Alleviation

[Hashstack Team]:

They are checking whether markets are supported or not now.

[Hashstack Team]:

marketData.minAmount is also updated in _updateMarketSupport(). After a market is added under the condition that its 200 days daily moving average drops below 1 billion we might consider removing that market.

LDH-15 | Unnecessary Function Parameter

Category	Severity	Location	Status
Gas Optimization	Informational	libraries/LibDiamond.sol: 2119, 2141	Resolved

Description

The `_facetAddress` parameter was not found to be necessary because the function would revert if it is not the Zero address. Therefore, it can be forced to always use `address(0)` instead of using the parameter.

Recommendation

We recommend to remove the `_facetAddress` parameter from the signature of the function.

Alleviation

[Hashstack Team]:

As per diamond standard EIP-2535, the function expects the _facetAddress argument in order to maintain consistency for all the upgradability functions.

https://github.com/mudgen/diamond-3-hardhat/blob/main/contracts/interfaces/IDiamondCut.sol#L13
https://eips.ethereum.org/EIPS/eip-2535

LDH-16 | _permissibleCDR() Logic

Category	Severity	Location	Status
Logical Issue	Informational	libraries/LibDiamond.sol: 1628~1632	Acknowledged

Description

In `_permissibleCDR()`, the max CDR is lowered to 2 instead of 3 if the market reserves after the loan decreases more than 25%. It is not explained in the white-paper, so could you please verify?

Recommendation

n/a

Alleviation

[Hashstack Team]:

Issue acknowledged. Changes have been reflected.

This technical detail is not required for whitepaper and thereby doesn't go into whitepaper.

LDH-17 | Undefined Fee Structure

Category	Severity	Location	Status
Inconsistency	Informational	libraries/LibDiamond.sol: 209~218	Acknowledged

Description

The whitepaper does not properly explain the fees structure; from page 13: `“The comptroller sets apy, apr as well as the fees for various protocol interactions. Initially, the fees structure”`.

Furthermore, the `LibDiamond.sol` file does not set fees, it just declares them as state variables. Only the `Comptroller.sol` mentions fees, but it was not in scope.

Recommendation

Update the whitepaper so it explains the fee structure, and include the `Comptroller.sol` file in future audits to review the fee implementation.

Alleviation

[Hashstack Team]:

Fees are updated in code and are getting set in deploy_all

Refer: https://github.com/0xHashstack/Open-contracts/blob/development/scripts/deploy_all.js#:~:text=///%20SET%20FEES%20IN,implemented%20in%20Comptroller%22)%3B

LDH-18 | State Variable `priceData` Is Not Regularly Updated

Category	Severity	Location	Status
Volatile Code	Informational	libraries/LibDiamond.sol: 1995~1996, 2001~2002	Resolved

Description

`priceData` is heavily used in `_getFairPrice()`, but it is only updated in `_fairPrice()`. However, being an `internal` function, `_fairPrice()` is never called

Recommendation

Recommend involving some oracle integrations, like Chainlink, which interface is already included in the repo, in the price feed implementations.

Alleviation

[Hashstack Team]:

We have removed _getFairPrice from the code base and no longer use it.

LHK-01 | Calling Functions `liquidation()` And `permissibleWithdrawal()` In The Same Block

Category	Severity	Location	Status
Volatile Code	Medium	Loan1.sol: 56~57, 61	Resolved

Description

Both functions `liquidation()` and `permissibleWithdrawal()` in contract Loan1 are wrapper functions for LibDiamond. In LibDiamond, function `_liquidation()` checks if the `loan.id` is valid, and function `_permissibleWithdraw()` checks if the loan/withdrawal amounts are valid. Although function `_accruedInterest()` is called and there is a check of the loan status in function `_accruedInterest()`, the status is never changed in these two functions. Our understanding is that there should be some status variables or time period variables to limit the users of calling these two functions in the same block.

Recommendation

Recommend tuning a threshold to be used as the minimal time period of calling these two functions, properly testing the threshold to make sure that it meets the design specifications, and documenting the threshold to reach the community consensus.

Alleviation

[Hashstack Team]:

Liquidation is under construction right now and does miss some functionalities.

[Hashstack Team]:

We are deleting loan records at places where the loan gets completed, and in all the functionalities the same check is in place (require(loan.id != 0)) for maintaining proper state.

This was missing in a function `withdrawPartialLoan` We have added the same in commit hash <3e4f2fb7613e90e06b8629747393a269e8a01af9>

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.

Mathematical Operations

Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

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

Control Flow

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

Volatile Code

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

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.

Coding Style

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

Inconsistency

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

Checksum Calculation Method

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

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

Disclaimer

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.

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.