CERTIK

Security Assessment

RiskHarbor

Apr 27th, 2021

Summary

This report has been prepared for RiskHarbor smart contracts, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, 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 given they are currently missing in the repository;
- Provide more comments per each function for readability, especially contracts are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	RiskHarbor
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/Risk-Harbor/RiskHarbor-Contracts
Commits	f942c8cdef7b4888abbb1410dedf924f48ae848e

Audit Summary

Delivery Date	Apr 27, 2021
Audit Methodology	Static Analysis, Manual Review, Testnet Deployment
Key Components	Consumer, Defaultector, GovToken, Shared, Underwriter, govTokenDistributor

Vulnerability Summary

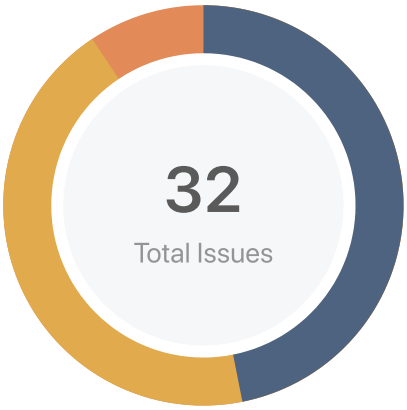
Total Issues	32
Critical	0
Major	3
Minor	14
Informational	15
Discussion	0

Audit Scope

ID	file	SHA256 Checksum
CCR	Consumer/Consumer.sol	75e2e7f3c89e959428cfe1c35c3d1047d14b3df95658aec442353c5a0a44bbb9
DCR	Consumer/Dripper.sol	61daff5aafbca55145f1d855f2d39b4406cf444f5d395fa3265fe81ae99acf06
IBA	Defaultector/Abstracts/IBase.sol	05fbc8b9839c38f6652c3fcd7a001cfb3d7f7d634ad703db2e51265821bded5a
ILM	Defaultector/Abstracts/ILendingMarket.sol	33ae7837550257e534840607737a2b6a2765eefa7bda808030dec6046e814c85
AWI	Defaultector/Implementations/AaveWrapper.sol	c0704db6fbdd2b9412b95834f4170e54fbac1a4a3cb5d75106801e3595a4f225
CWI	Defaultector/Implementations/CompoundWrapper.sol	39bf3e2ec01253bca634597b739a1f15b1bdcef5a3845f725eacfe5491870cc4
YWI	Defaultector/Implementations/YearnWrapper.sol	699e0c20cc889ad233ded12cccae60d05b05e7eedd30b18a3e8bea7710404692
IAI	Defaultector/Interfaces/IAave.sol	48f81892f507bf9782f548814417389d16a7313ea4aa2dceafd63e0d80404ea1
ICI	Defaultector/Interfaces/ICompound.sol	63c41877be24a24b0dcbcde03caaf019cfc42f0191fc8ede6aba3ccf5f217b04
IYI	Defaultector/Interfaces/IYearn.sol	0cfd0e966ffb574c37f408d8b8cdee5f47a397cae7facdc92211a833979df680
RHG	GovToken/RiskHarbor.sol	516146b085208d44549eece4203239495d21e688043c6f29971d98a2767f0944
ICR	Interfaces/IConsumer.sol	1173f6e2dd265581ca60e550c432d2ef5e05b5b935984dd83c16b26296768a88
IMD	Interfaces/IMerkleDistributor.sol	b14e48256a8a1425b05870646a2be82abea1261054d846c84db8ccd6d2fd2a9d
IUI	Interfaces/IUnderwriter.sol	88f3fa63c596a1a0b9bd6ca7c77651f14776ecf026a81ec4d8a25389b7f5ede5
ERS	Shared/ErrorReporter.sol	4676f0aec3f7326ac03e5bfe7b6989e4dce460f229f344b23309fb111e3a943a
SSR	Shared/Shared.sol	350e04608524ec4a31b8327accca727b7c6d671baff0c64304626c28ac7f16de
SSH	Storage/Storage.sol	9e92d8007c5e0bedb45f593f77b1c093774019f2c24c71f39a041a1eea82aa0d
CUR	Underwriter/Claims.sol	4b885ad938d701a07057cf8854607417773138969957a62a48d9ece62540decba
CUH	Underwriter/Conversion.sol	4686d887650a5eb8a1d5bf44c07fa4d211318923e7453b68f8fd1dd3b4412a53

ID	file	SHA256 Checksum
MSU	Underwriter/MarketStatus.sol	c78de0dd280e8603497714f80b193102e04abf323f027d11690614da330901ee
SUR	Underwriter/Shares.sol	8ebc5bec2a40bbda4f5cb69f70a3cde7be020fcd1502904e139f42355b5a2823
UUR	Underwriter/Underwriter.sol	d05efc6ba93c79a216ccfd0a30b72b37514cf2407178a4b8870a47e7e29f17a1
UUH	Underwriter/Utilization.sol	6798af9950c9decc2b72be89f4f7bb41fba82850bd1254667a8a1ec595ee5332
MDT	govTokenDistributor/MerkleDistributor.sol	7972fe0bc8bf99cdcb635b0532143c85a509fbb64d9190e440c11b7259bf69ab

Findings



Critical	**0** (0.00%)
Major	**3** (9.38%)
Minor	**14** (43.75%)
Informational	**15** (46.88%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
GLOBAL-01	`marketStatus` Never Set to `Status.OPEN`	Logical Issue	Minor	Acknowledged
CCR-01	Gov Token Transferring to `underwriter` instead of admin	Control Flow	Major	Resolved
CCR-02	Privileged Ownership	**Control Flow, Centralization / Privilege**	**Minor**	**Acknowledged**
CCR-03	Missing Checks for Scenario Described in Comments	Logical Issue	Minor	Resolved
CCR-04	Runtime State Variables Missing Event Emitting	Coding Style	Informational	Resolved
CCR-05	Missing Interface Inheritance	Volatile Code	Minor	Resolved
CCR-06	Missing Zero-Address Validation	Volatile Code	Minor	Resolved
CUR-01	Runtime State Variables Missing Event Emitting	Coding Style	Informational	Resolved
CWI-01	Wrapper Contract Coverage	Business Model	Major	Acknowledged
CWI-02	ERC20 Function Return Value Ignored	Language Specific	Minor	Resolved
CWI-03	Unnecessary Return Value Declare	Volatile Code	Informational	Resolved

ID	Title	Category	Severity	Status
DCR-01	Declaring `_drip()` as Modifier instead of Function	Coding Style	Informational	Resolved
DCR-02	Runtime State Variables Missing Event Emitting	Coding Style	Informational	Resolved
ERS-01	Compiler Warning on Restricting Function as `pure`	Coding Style, Compiler Error	Informational	Resolved
MDT-01	Missing Zero-Address Validation	Volatile Code	Minor	Resolved
MSU-01	Modifier Never Used	Coding Style	Informational	Resolved
MSU-02	Runtime State Variables Missing Event Emitting	Coding Style	Informational	Resolved
RHG-01	Timestamp Dependence	Volatile Code	Informational	Acknowledged
RHG-02	Missing Zero-Address Validation	Volatile Code	Minor	Resolved
SSH-01	Runtime State Variables Missing Event Emitting	Coding Style	Informational	Resolved
SUR-01	ERC20 Function Return Value Ignored	Language Specific	Minor	Resolved
SUR-02	Missing Pre-Declared Return Value	Volatile Code	Minor	Resolved
SUR-03	Coding Style Inconsistency	Coding Style, Inconsistency	Informational	Acknowledged
UUH-01	`public` Functions Could Be Declared `external`	Gas Optimization	Informational	Resolved
UUR-01	Privileged Ownership	**Control Flow, Centralization / Privilege**	**Minor**	**Acknowledged**
UUR-02	Declaring `_drip()` as Modifier instead of Function	Coding Style	Informational	Resolved
UUR-03	ERC20 Function Return Value Ignored	Language Specific	Minor	Resolved

ID	Title	Category	Severity	Status
UUR-04	Runtime State Variables Missing Event Emitting	Coding Style	Informational	Resolved
UUR-05	Missing Zero-Address Validation	Volatile Code	Minor	Resolved
YWI-01	Wrapper Contract Coverage	Business Model	Major	Acknowledged
YWI-02	ERC20 Function Return Value Ignored	Language Specific	Minor	Resolved
YWI-03	Unnecessary Return Value Declare	Volatile Code	Informational	Resolved

GLOBAL-01 | `marketStatus` Never Set to `Status.OPEN`

Category	Severity	Location	Status
Logical Issue	Minor	Global	Acknowledged

Description

We noticed that the `marketStatus` would be set to `HACKED` when claim checks failed, and it would be set to `CLOSED` when expiration time reached for `OPEN` market or grace period reached for `HACKED` market.

However, it is still unclear when the `marketStatus` would be set back to `OPEN`. Our assumption is that the `Storage` contract is a long-running contract, and it is not designed to be initialized multiple times for each market.

Alleviation

According to Risk Harbor team:

Currently, each set (underwriter and consumer contracts) isn't meant for reuse. In future versions, we hope to offer perpetual insurance where sets of insurance contracts are meant to last forever.

The code implementation meets requirement.

CCR-01 | Gov Token Transferring to `underwriter` instead of admin

Category	Severity	Location	Status
Control Flow	Major	Consumer/Consumer.sol: 145~146	Resolved

Description

In function `emergencyDump()`, according to the comments, it seems both gov token and credit token should be transferred to an admin address instead of `underwriter`.

Alleviation

Fixed in commit hash `444e547effdbe5c5409ba52697c7f9a8aabe2d94`.

CCR-02 | Privileged Ownership

Category	Severity	Location	Status
Control Flow, Centralization / Privilege	Minor	Consumer/Consumer.sol: 111~112, 143~144	Acknowledged

Description

The owner address has the ability to call functions `setPricePercent()`, `emergencyDump()`, `changeLimit()`, `upgradeDefaultector()` and `closeMarket()` without obtaining the consensus of the community.

Recommendation

Renounce ownership when it is the right timing; or gradually migrate to a timelock plus multisig governing procedure and let the community to monitor in respect of transparency considerations.

Alleviation

Acknowledged by RiskHarbor Team, "Will transfer from admin -> gnosis multisig -> governance as time goes on. Single admin control in the beginning to allow for rapid response in the early days."

CCR-03 | Missing Checks for Scenario Described in Comments

Category	Severity	Location	Status
Logical Issue	Minor	Consumer/Consumer.sol: 130~132	Resolved

Description

The comment of `dump()` claims that (the function) "Needed when a claim succeeds and the grace period ends". It seems that the logic should be "if and only if when a claim succeeds...". Therefore, there should be `require` checks or revert calls to avoid malicious underwriters calling `dump()` whenever they want.

Alleviation

Fixed in commit hash `444e547effdbe5c5409ba52697c7f9a8aabe2d94`.

CCR-04 | Runtime State Variables Missing Event Emitting

Category	Severity	Location	Status
Coding Style	Informational	Consumer/Consumer.sol: 113~114	Resolved

Description

In contract `Storage`, there are a bunch of state variables declared. According to the comments, these state variables can be classified to two types: `Constructor Params` and `Runtime: xxx`. Note that `price` can also be changed during runtime in function `setPricePercent()` of contract `Consumer`.

Recommend emitting events, for all the critical state variables that are possible to be changed during runtime.

Alleviation

Fixed in commit hash `f942c8cdef7b4888abbb1410dedf924f48ae848e`.

CCR-05 | Missing Interface Inheritance

Category	Severity	Location	Status
Volatile Code	Minor	Consumer/Consumer.sol: 23~30	Resolved

Description

Contract `Consumer` should inherit from interface `IConsumer`.

Alleviation

Fixed in commit hash `f942c8cdef7b4888abbb1410dedf924f48ae848e`. According to the RiskHarbor Team, "Had to make Dripper.sol an abstract contract since it wouldn't implement all the functions of IConsumer.sol. Made getPricePercent external alongside appending internal calls to it with `this.`"

CCR-06 | Missing Zero-Address Validation

Category	Severity	Location	Status
Volatile Code	Minor	Consumer/Consumer.sol: 55~63	Resolved

Description

In functions `constructor()` and `initialize()`, several token/wallet/etc. addresses have their value assigned. However, there are no address checks to ensure the addresses are not `address(0)`.

Recommendation

Recommend adding zero-address checks to revert invalid contract deployment.

Alleviation

Fixed in commit hash `f942c8cdef7b4888abbb1410dedf924f48ae848e`.

CUR-01 | Runtime State Variables Missing Event Emitting

Category	Severity	Location	Status
Coding Style	Informational	Underwriter/Claims.sol: 47~48, 44~45	Resolved

Description

In contract `Storage`, there are a bunch of state variables declared. According to the comments, these state variables can be classified to two types: `Constructor Params` and `Runtime: xxx`. Note that `price` can also be changed during runtime in function `setPricePercent()` of contract `Consumer`.

Recommend emitting events, for all the critical state variables that are possible to be changed during runtime.

Alleviation

Fixed in commit hash `f942c8cdef7b4888abbb1410dedf924f48ae848e`.

CWI-01 | Wrapper Contract Coverage

Category	Severity	Location	Status
Business Model	Major	Defaultector/Implementations/CompoundWrapper.sol: 1	Acknowledged

Description

From our current understanding, function `checkRedeemability()` is actually checking if the credit token can be withdrawn/redeemed. This check could help on cases of either the protocol of credit token rug pulled or the protocol community pause/lock the token transfer because of some hacks or crisis. For the later case, it is possible that the withdraw/redeem failed in the certain amount of time, and then probably the credit token transfer is unpaused/unlocked by the protocol community.

Furthermore, for the case that the wrapper contracts are blacklisted, the whole functionality would fail.

Alleviation

According to Risk Harbor team:

Later down the line, we will offer policyholders multiple options for coverage to buy. One of these addons will be covering pause events. We will just call the insured contract's paused function bool to see if it's paused. Currently, if it's paused it will payout. Compound doesn't have a pause function as far as I know. Yearn's pausability is unknown.

For the case of being blacklisted:

In this case, we would have to use the admin function to force close the market and pay back the remaining premiums.

CWI-02 | ERC20 Function Return Value Ignored

Category	Severity	Location	Status
Language Specific	Minor	Defaultector/Implementations/CompoundWrapper.sol: 67~68	Resolved

Description

In contracts `Shares`, `Underwriter`, `CompoundWrapper` and `YearnWrapper`, there are ERC20 function calls of `transfer` and `transferFrom` without the return value well handled.

According to EIP-20:

Callers MUST handle `false` from `returns (bool success)`. Callers MUST NOT assume that `false` is never returned!

Recommendation

Recommend follow the instructions of standard ERC20 interface.

Alleviation

Fixed in commit hash `f942c8cdef7b4888abbb1410dedf924f48ae848e`, according to the RiskHarbor Team, "Using SafeERC20 transfers to handle return of false."

CWI-03 | Unnecessary Return Value Declare

Category	Severity	Location	Status
Volatile Code	Informational	Defaultector/Implementations/CompoundWrapper.sol: 79~80	Resolved

Description

Function `attemptFailedWithdraw()` in contracts `CompoundWrapper` and `YearnWrapper` has declared a return value of `uint256`. However, the function would always `revert` and never return any variable.

Recommendation

Recommend match the function declaration and the actual behavior.

Alleviation

Fixed in commit hash `f942c8cdef7b4888abbb1410dedf924f48ae848e`, according to the RiskHarbor Team, "removing return (uint256) from all defaultector wrappers."

DCR-01 | Declaring `_drip()` as Modifier instead of Function

Category	Severity	Location	Status
Coding Style	Informational	Consumer/Dripper.sol: 17~18, 35~36	Resolved

Description

In contract `Dripper`, the modifier `_drip()` is only used in `external` function `drip()`, which is an empty function just to call the modifier. Then after searching all contracts, we found that the `external` call of `drip()` is from modifier `drip` in contract `Underwriter`.

Therefore, it seems in contract `Dripper`, the modifier `_drip()` could be declared as a function instead of a modifier. Just would like to learn are there any special reasons of declaring it as a modifier and wrapping it afterwards?

Alleviation

Fixed in commit hash `f942c8cdef7b4888abbb1410dedf924f48ae848e` by removing empty function, according to RiskHarbor Team, "The previous rationale was that Consumer functions like Purchase would also call drip. We decided to shift the gas burden to the underwriters. Also added onlyUnderwriter modifier to the drip function to prevent outsiders from potentially exploiting it."

DCR-02 | Runtime State Variables Missing Event Emitting

Category	Severity	Location	Status
Coding Style	Informational	Consumer/Dripper.sol: 24~25	Resolved

Description

In contract `Storage`, there are a bunch of state variables declared. According to the comments, these state variables can be classified to two types: `Constructor Params` and `Runtime: xxx`. Note that `price` can also be changed during runtime in function `setPricePercent()` of contract `Consumer`.

Recommend emitting events, for all the critical state variables that are possible to be changed during runtime.

Alleviation

Fixed in commit hash `f942c8cdef7b4888abbb1410dedf924f48ae848e`.

ERS-01 | Compiler Warning on Restricting Function as `pure`

Category	Severity	Location	Status
Coding Style, Compiler Error	Informational	Shared/ErrorReporter.sol: 34~35	Resolved

Description

Compiled with Solidity version `0.7.0`, there are several compiler warning saying that "Warning: Function state mutability can be restricted to pure". Excluding those unfinished/todo functions in `xxxWrapper`, the only one needs to be take care of is function `error()` of contract `ErrorReporter`. Note that if a function does not read storage state, it can be declared as `pure`, according to Solidity Documentation.

Alleviation

Fixed in commit hash `f942c8cdef7b4888abbb1410dedf924f48ae848e`.

MDT-01 | Missing Zero-Address Validation

Category	Severity	Location	Status
Volatile Code	Minor	govTokenDistributor/MerkleDistributor.sol: 16~17	Resolved

Description

In functions `constructor()` and `initialize()`, several token/wallet/etc. addresses have their value assigned. However, there are no address checks to ensure the addresses are not `address(0)`.

Recommendation

Recommend adding zero-address checks to revert invalid contract deployment.

Alleviation

Fixed in commit hash `f942c8cdef7b4888abbb1410dedf924f48ae848e`.

MSU-01 | Modifier Never Used

Category	Severity	Location	Status
Coding Style	Informational	Underwriter/MarketStatus.sol: 47~48	Resolved

Description

Currently the modifier `_atStatus2` is never used. Given the fact that there are now three types of status `{OPEN, HACKED, CLOSED}`, we would like to learn what will the use cases for `_atStatus2`, and will there be more types of status, other than the current three types?

Alleviation

Fixed in commit hash `f942c8cdef7b4888abbb1410dedf924f48ae848e`, according to the RiskHarbor Team, "the rationale previously was to have 4 market states meaning that we can't just do a single notAtStatus check. But since we only have 3 now, we can do a notAtStatus check."

MSU-02 | Runtime State Variables Missing Event Emitting

Category	Severity	Location	Status
Coding Style	Informational	Underwriter/MarketStatus.sol: 17~18, 24~25	Resolved

Description

In contract `Storage`, there are a bunch of state variables declared. According to the comments, these state variables can be classified to two types: `Constructor Params` and `Runtime: xxx`. Note that `price` can also be changed during runtime in function `setPricePercent()` of contract `Consumer`.

Recommend emitting events, for all the critical state variables that are possible to be changed during runtime.

Alleviation

Fixed in commit hash `f942c8cdef7b4888abbb1410dedf924f48ae848e`.

RHG-01 | Timestamp Dependence

Category	Severity	Location	Status
Volatile Code	Informational	GovToken/RiskHarbor.sol: 42~46	Acknowledged

Description

In function `release()`, there is a `require` statement checking the vesting is complete using `block.timestamp`. Note the block time on testnet and mainnet Ethereum are different. Please understand the security risk level and trade-off of using `block.timestamp` or alias `now` as one of core factors in the contract.

Recommendation

Correct use of 15-second rule to minimize the impact caused by timestamp variance

Alleviation

Acknowledged by the RiskHarbor Team: Per https://ethereum.stackexchange.com/questions/6795/is-block-timestamp-safe-for-longer-time-periods, since it's for checking if a year has passed this shouldn't be much of an issue since max time inaccuracy is 900 seconds.

RHG-02 | Missing Zero-Address Validation

Category	Severity	Location	Status
Volatile Code	Minor	GovToken/RiskHarbor.sol: 36~37	Resolved

Description

In functions `constructor()` and `initialize()`, several token/wallet/etc. addresses have their value assigned. However, there are no address checks to ensure the addresses are not `address(0)`.

Recommendation

Recommend adding zero-address checks to revert invalid contract deployment.

Alleviation

Fixed in commit hash `f942c8cdef7b4888abbb1410dedf924f48ae848e`.

SSH-01 | Runtime State Variables Missing Event Emitting

Category	Severity	Location	Status
Coding Style	Informational	Storage/Storage.sol: 1	Resolved

Description

In contract `Storage`, there are a bunch of state variables declared. According to the comments, these state variables can be classified to two types: `Constructor Params` and `Runtime: xxx`. Note that `price` can also be changed during runtime in function `setPricePercent()` of contract `Consumer`.

Recommend emitting events, for all the critical state variables that are possible to be changed during runtime.

Alleviation

Fixed in commit hash `f942c8cdef7b4888abbb1410dedf924f48ae848e`.

SUR-01 | ERC20 Function Return Value Ignored

Category	Severity	Location	Status
Language Specific	Minor	Underwriter/Shares.sol: 78~79, 127~128, 129~130, 131~132	Resolved

Description

In contracts `Shares`, `Underwriter`, `CompoundWrapper` and `YearnWrapper`, there are ERC20 function calls of `transfer` and `transferFrom` without the return value well handled.

According to EIP-20:

Callers MUST handle `false` from `returns (bool success)`. Callers MUST NOT assume that `false` is never returned!

Recommendation

Recommend follow the instructions of standard ERC20 interface.

Alleviation

Fixed in commit hash `f942c8cdef7b4888abbb1410dedf924f48ae848e`, according to the RiskHarbor Team, "Using SafeERC20 transfers to handle return of false."

SUR-02 | Missing Pre-Declared Return Value

Category	Severity	Location	Status
Volatile Code	Minor	Underwriter/Shares.sol: 56~79	Resolved

Description

In L56, function `buyShares` declares a `uint256` number to be returned. However, the function did not return any value.

In addition, `Shares.buyShares()` is called in function `deposit()` of contract `Underwriter`. If no values are returned from `Shares.buyShares`, the event `PurchasedShares` will always emit an empty value for `shares`.

```
...
uint256 shares = Shares.buyShares(_amount);
...
emit PurchasedShares(msg.sender, _amount, shares);
```

Alleviation

Fixed in commit hash `444e547effdbe5c5409ba52697c7f9a8aabe2d94`.

SUR-03 | Coding Style Inconsistency

Category	Severity	Location	Status
Coding Style, Inconsistency	Informational	Underwriter/Shares.sol: 81~82	Acknowledged

Description

In L82 of function `buyShares()`, there is an `assert` statement. It is recommended using `require` statement to check conditions, since a message string can be provided. However, if the case is limited to internal error checking, `assert` statement should be also good.

```
// Make sure token transfer didn't silently fail
assert(currToken.balanceOf(address(this)) == prevBal.add(_amount));
```

Alleviation

Acknowledged by the RiskHarbor Team:

We are following the pattern of using assert to check effects per commonly used standards

UUH-01 | `public` Functions Could Be Declared `external`

Category	Severity	Location	Status
Gas Optimization	Informational	Underwriter/Utilization.sol: 26~29, 40~43	Resolved

Description

Functions `getTotalCapacityInCredit()` and `getUtilizedAmountCredit()` are never used in other contracts. Declaring functions as `external` could help save gas.

Alleviation

Fixed in commit hash `f942c8cdef7b4888abbb1410dedf924f48ae848e`.

UUR-01 | Privileged Ownership

Category	Severity	Location	Status
Control Flow, Centralization / Privilege	Minor	Underwriter/Underwriter.sol: 222~223, 236~237, 242~243, 247~248	Acknowledged

Description

The owner address has the ability to call functions `setPricePercent()`, `emergencyDump()`, `changeLimit()`, `upgradeDefaultector()` and `closeMarket()` without obtaining the consensus of the community.

Recommendation

Renounce ownership when it is the right timing; or gradually migrate to a timelock plus multisig governing procedure and let the community to monitor in respect of transparency considerations.

Alleviation

Acknowledged by RiskHarbor Team, "Will transfer from admin -> gnosis multisig -> governance as time goes on. Single admin control in the beginning to allow for rapid response in the early days."

UUR-02 | Declaring `_drip()` as Modifier instead of Function

Category	Severity	Location	Status
Coding Style	Informational	Underwriter/Underwriter.sol: 214~215	Resolved

Description

In contract `Dripper`, the modifier `_drip()` is only used in `external` function `drip()`, which is an empty function just to call the modifier. Then after searching all contracts, we found that the `external` call of `drip()` is from modifier `drip` in contract `Underwriter`.

Therefore, it seems in contract `Dripper`, the modifier `_drip()` could be declared as a function instead of a modifier. Just would like to learn are there any special reasons of declaring it as a modifier and wrapping it afterwards?

Alleviation

Fixed in commit hash `f942c8cdef7b4888abbb1410dedf924f48ae848e` by removing empty function, according to RiskHarbor Team, "The previous rationale was that Consumer functions like Purchase would also call drip. We decided to shift the gas burden to the underwriters. Also added onlyUnderwriter modifier to the drip function to prevent outsiders from potentially exploiting it."

UUR-03 | ERC20 Function Return Value Ignored

Category	Severity	Location	Status
Language Specific	Minor	Underwriter/Underwriter.sol: 208~209, 249~253, 253~257	Resolved

Description

In contracts `Shares`, `Underwriter`, `CompoundWrapper` and `YearnWrapper`, there are ERC20 function calls of `transfer` and `transferFrom` without the return value well handled.

According to EIP-20:

Callers MUST handle `false` from `returns (bool success)`. Callers MUST NOT assume that `false` is never returned!

Recommendation

Recommend follow the instructions of standard ERC20 interface.

Alleviation

Fixed in commit hash `f942c8cdef7b4888abbb1410dedf924f48ae848e`, according to the RiskHarbor Team, "Using SafeERC20 transfers to handle return of false."

UUR-04 | Runtime State Variables Missing Event Emitting

Category	Severity	Location	Status
Coding Style	Informational	Underwriter/Underwriter.sol: 238~239, 223~224	Resolved

Description

In contract `Storage`, there are a bunch of state variables declared. According to the comments, these state variables can be classified to two types: `Constructor Params` and `Runtime: xxx`. Note that `price` can also be changed during runtime in function `setPricePercent()` of contract `Consumer`.

Recommend emitting events, for all the critical state variables that are possible to be changed during runtime.

Alleviation

Fixed in commit hash `f942c8cdef7b4888abbb1410dedf924f48ae848e`.

UUR-05 | Missing Zero-Address Validation

Category	Severity	Location	Status
Volatile Code	Minor	Underwriter/Underwriter.sol: 78~88	Resolved

Description

In functions `constructor()` and `initialize()`, several token/wallet/etc. addresses have their value assigned. However, there are no address checks to ensure the addresses are not `address(0)`.

Recommendation

Recommend adding zero-address checks to revert invalid contract deployment.

Alleviation

Fixed in commit hash `f942c8cdef7b4888abbb1410dedf924f48ae848e`.

YWI-01 | Wrapper Contract Coverage

Category	Severity	Location	Status
Business Model	Major	Defaultector/Implementations/YearnWrapper.sol: 1	Acknowledged

Description

From our current understanding, function `checkRedeemability()` is actually checking if the credit token can be withdrawn/redeemed. This check could help on cases of either the protocol of credit token rug pulled or the protocol community pause/lock the token transfer because of some hacks or crisis. For the later case, it is possible that the withdraw/redeem failed in the certain amount of time, and then probably the credit token transfer is unpaused/unlocked by the protocol community.

Furthermore, for the case that the wrapper contracts are blacklisted, the whole functionality would fail.

Alleviation

According to Risk Harbor team:

Later down the line, we will offer policyholders multiple options for coverage to buy. One of these addons will be covering pause events. We will just call the insured contract's paused function bool to see if it's paused. Currently, if it's paused it will payout. Compound doesn't have a pause function as far as I know. Yearn's pausability is unknown.

For the case of being blacklisted:

In this case, we would have to use the admin function to force close the market and pay back the remaining premiums.

YWI-02 | ERC20 Function Return Value Ignored

Category	Severity	Location	Status
Language Specific	Minor	Defaultector/Implementations/YearnWrapper.sol: 48~49	Resolved

Description

In contracts `Shares`, `Underwriter`, `CompoundWrapper` and `YearnWrapper`, there are ERC20 function calls of `transfer` and `transferFrom` without the return value well handled.

According to EIP-20:

Callers MUST handle `false` from `returns (bool success)`. Callers MUST NOT assume that `false` is never returned!

Recommendation

Recommend follow the instructions of standard ERC20 interface.

Alleviation

Fixed in commit hash `f942c8cdef7b4888abbb1410dedf924f48ae848e`, according to the RiskHarbor Team, "Using SafeERC20 transfers to handle return of false."

YWI-03 | Unnecessary Return Value Declare

Category	Severity	Location	Status
Volatile Code	Informational	Defaultector/Implementations/YearnWrapper.sol: 60~61	Resolved

Description

Function `attemptFailedWithdraw()` in contracts `CompoundWrapper` and `YearnWrapper` has declared a return value of `uint256`. However, the function would always `revert` and never return any variable.

Recommendation

Recommend match the function declaration and the actual behavior.

Alleviation

Fixed in commit hash `f942c8cdef7b4888abbb1410dedf924f48ae848e`, according to the RiskHarbor Team, "removing return (uint256) from all defaultector wrappers."

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.

Mathematical Operations

Mathematical Operation 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.

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 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 without CertiK's prior written consent.

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.

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.