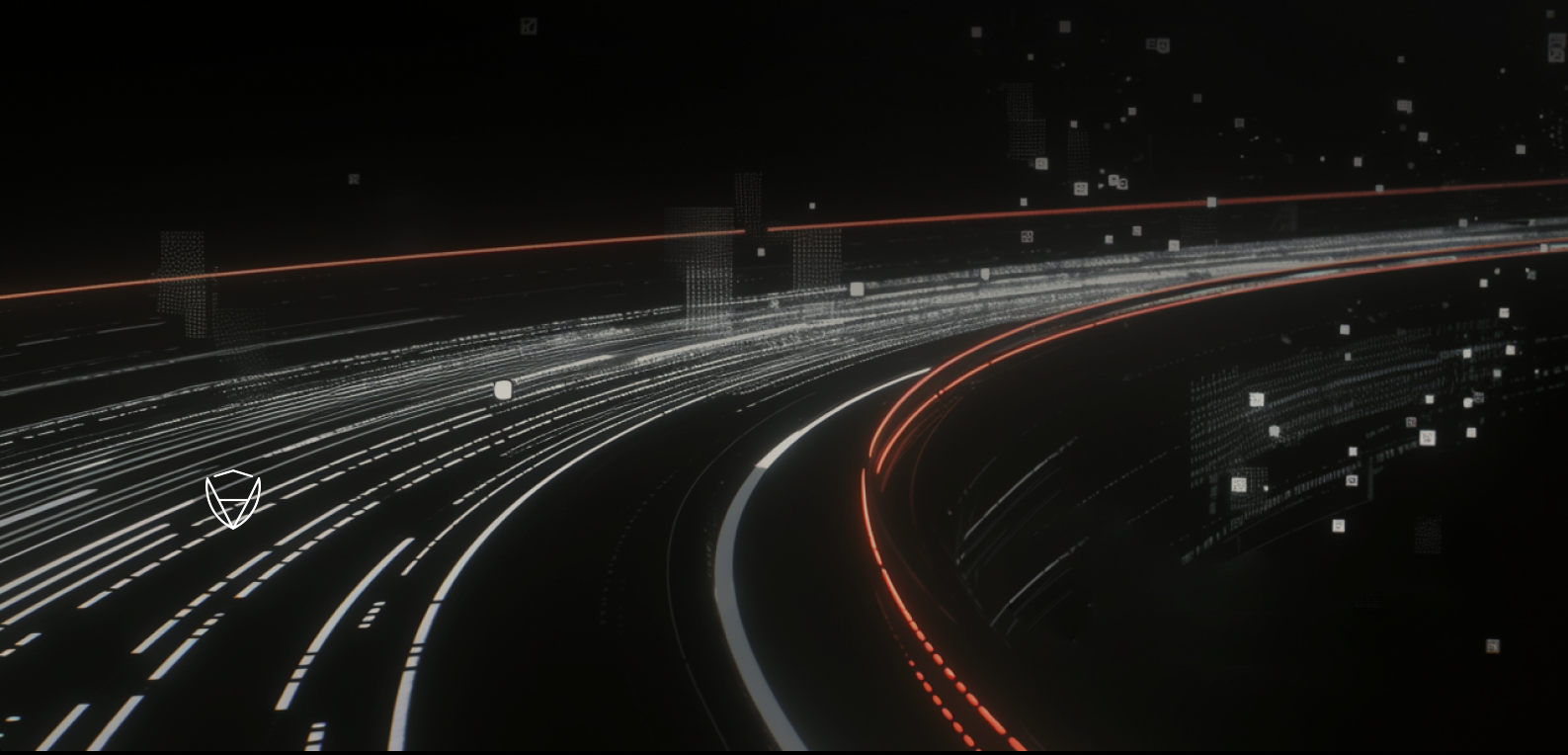
CERTIK

Security Assessment

Hilter

CertiK Assessed on July 30, 2025



CertiK Assessed on July 30, 2025

Hilter

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

Executive Summary

TYPES	ECOSYSTEM	METHODS
DeFi	EVM Compatible	Formal Verification, Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 07/30/2025	N/A

CODEBASE

https://gitlab.com/hilterltd-group/hilter-defi/packages/contracts/src

View All in Codebase Page

COMMITS

- da9877c0823663a5fca5f75f3a09b84e050dab8e
- 9b86f2af4630281730 79c7180e231c6f14325c54

View All in Codebase Page

Vulnerability Summary

6	4	0	0	2	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined

0 Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
0 Major		Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
2 Medium	2 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
3 Minor	1 Resolved, 2 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
1 Informational	1 Resolved	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | HILTER

CODEBASE | HILTER

Repository

https://gitlab.com/hilterltd-group/hilter-defi/packages/contracts/src

Commit

- da9877c0823663a5fca5f75f3a09b84e050dab8e
- 9b86f2af463028173079c7180e231c6f14325c54

AUDIT SCOPE | HILTER

30 files audited ● 2 files with Acknowledged findings ● 28 files without findings

ID	Repo	File	SHA256 Checksum
HBV	hilterltd-group/ hilter-defi	base/HilterlBaseVault.sol	746d77413ed9e406ea61e32cde5ae555ff23c 112dfd40dc9b73fd6f7f761b7a5
UVB	hilterltd-group/ hilter-defi	vaults/UpsideVault.sol	24b73c346770cd0b044ee45c0c8401725175 72261d086e73cd474608569c4d44
HKY	hilterltd-group/ hilter-defi	HilterKYCProvider.sol	1b9228c7f2e6b2289ae5b18cd64373796dc4 adf3910bb1ce838f8680ba6329b4
HFY	hilterltd-group/ hilter-defi	HilterFixedYieldVaultWithUpsid e.sol	b020e54eb3fb146c8e9d997177a1197b8641 77693707b66c04dbd567a11d620a
HFV	hilterltd-group/ hilter-defi	HilterFixedYieldVault.sol	07f7969bf5a99cc944bf92506659663634e7f7 a1e9964ada22bd9a395c0b8140
MVB	hilterltd-group/ hilter-defi	extensions/MaturityVault.sol	24619b65444c6bc19e69350145681b8e71b0 341b52aaff45fac8d032c2adb864
HFF	hilterltd-group/ hilter-defi	factories/HilterFixedYieldVaultFa ctory.sol	7b8658de2db4e7c17dd01bf149ad7b25ed71 a098037fc4e68b2a33bb195330a6
HUV	hilterltd-group/ hilter-defi	factories/HilterUpsideVaultFactory. sol	6eb9b98e837fb1f2d7c61d34ab354e7c40135 8f860003286531b31c2163e581c
HVF	hilterltd-group/ hilter-defi	factories/HilterlVaultFactory.sol	be881491aa86d2d203146ac91dbd92539a09 851b94086d564267fe55dbeef729
MCP	hilterltd-group/ hilter-defi	plugins/MaxCapPlug.sol	6a088c4f87a285941f97d1267b848fb0c5cd2 5f89f9fa2a1b4341514430b7035
WPI	hilterltd-group/ hilter-defi	plugins/WhitelistPlugIn.sol	c3e4faaa4bb6ef3c05abdb6480ffad6bfc04e4 d9b545bf3213cc2f53b4aaa370
WIN	hilterltd-group/ hilter-defi	plugins/WindowPlugIn.sol	c79b09843e9b2038710f7a33fa56b8810533d 26a9262a597441ed1d05c7fd775
FYV	hilterltd-group/ hilter-defi	vaults/FixedYieldVault.sol	c7aa8297b9288698541a92a95197cecfef8f8 ddcc7dbee8d11cf6d3268c73db7

ID	Repo	File	SHA256 Checksum
IHB	hilterltd-group/hilter-defi	interface/IHilter.sol	ce476844f5b7b27820becf49288f263d8303468a3710b166dc746880dfab0fe1
IKY	hilterltd-group/hilter-defi	interface/IKYCProvider.sol	6bdbdec57340f10c318c08f47c9ca20b77290268de80ecc6e9cc9aaf71751bac
HRE	hilterltd-group/hilter-defi	base/HilterBaseVault.sol	95d97f2861995ce59308b87f0a5e8b300197c64f660e998bc1d0f1f73dba8182
MVU	hilterltd-group/hilter-defi	extensions/MaturityVault.sol	24619b65444c6bc19e69350145681b8e71b0341b52aaff45fac8d032c2adb864
HYF	hilterltd-group/hilter-defi	factories/HilterFixedYieldVaultFactory.sol	7b8658de2db4e7c17dd01bf149ad7b25ed71a098037fc4e68b2a33bb195330a6
HUF	hilterltd-group/hilter-defi	factories/HilterUpsideVaultFactory.sol	6eb9b98e837fb1f2d7c61d34ab354e7c401358f860003286531b31c2163e581c
HRD	hilterltd-group/hilter-defi	factories/HilterVaultFactory.sol	be881491aa86d2d203146ac91dbd92539a09851b94086d564267fe55dbeef729
IHU	hilterltd-group/hilter-defi	interface/IHilter.sol	ce476844f5b7b27820becf49288f263d8303468a3710b166dc746880dfab0fe1
IKC	hilterltd-group/hilter-defi	interface/IKYCProvider.sol	6bdbdec57340f10c318c08f47c9ca20b77290268de80ecc6e9cc9aaf71751bac
MAX	hilterltd-group/hilter-defi	plugins/MaxCapPlug.sol	6a088c4f87a285941f97d1267b848fb0c5cd25f89f9fa2a1b4341514430b7035
WHI	hilterltd-group/hilter-defi	plugins/WhitelistPlugIn.sol	c3e4faaa4bb6ef3c05abdb6480ffad6bfc04e4d9b545bf3213cc2f53b4aaa370
WID	hilterltd-group/hilter-defi	plugins/WindowPlugIn.sol	c79b09843e9b2038710f7a33fa56b8810533d26a9262a597441ed1d05c7fd775
FIX	hilterltd-group/hilter-defi	vaults/FixedYieldVault.sol	57b1a685c6050ac8ac95b0f2401145afa765ab8e9d243dbfead354d9d4c5bbfc
UVU	hilterltd-group/hilter-defi	vaults/UpsideVault.sol	3c3f66d642d7c9837285edb18400bf0581cb9ae019df7edd8b292515317cb58e
HYV	hilterltd-group/hilter-defi	HilterFixedYieldVault.sol	07f7969bf5a99cc944bf92506659663634e7f7a1e9964ada22bd9a395c0b8140

ID	Repo	File	SHA256 Checksum
HFW	hilterltd-group/ hilter-defi	HilterFixedYieldVaultWithUpside.sol	b020e54eb3fb146c8e9d997177a1197b864177693707b66c04dbd567a11d620a
HKC	hilterltd-group/ hilter-defi	HilterKYCProvider.sol	1b9228c7f2e6b2289ae5b18cd64373796dc4adf3910bb1ce838f8680ba6329b4

APPROACH & METHODS | HILTER

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

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

FINDINGS | HILTER

6	0	0	2	3	1
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Hilter. Through this audit, we have uncovered 8 issues ranging from different severity levels. Utilizing the techniques of Static Analysis & Manual Review to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
SRC-02	Potential Asset Lock In `FixedYieldVault` And `UpsideVault` Contracts	Logical Issue	Medium	Resolved
UVB-01	Inconsistent Use Of `Pausable` Contract In `UpsideVault`	Access Control	Medium	Resolved
HBV-01	Tokens Transferred To Vault Will Be Locked	Design Issue	Minor	Resolved
SRC-01	Incompatibility With Deflationary Tokens (Non-Standard ERC20 Token)	Volatile Code	Minor	Acknowledged
SRC-03	Third-Party Dependencies	Volatile Code	Minor	Acknowledged
UVB-03	Shadowing State Variable	Coding Style	Informational	Resolved

SRC-02 POTENTIAL ASSET LOCK IN `FixedYieldVault` AND `UpsideVault` CONTRACTS

Category	Severity	Location	Status
Logical Issue	Medium	base/HilterBaseVault.sol (da9877): 103; vaults/UpsideVault.sol (da9877): 92	Resolved

Description

The `FixedYieldVault` and `UpsideVault` contracts use the `totalAssetDeposited` variable to track the total assets deposited into the vault. This variable is adjusted upward during deposits and downward during withdrawals. However, if assets are directly transferred into the vault without using the `_deposit()` function (e.g., through donations or direct transfers), these assets are not recorded in `totalAssetDeposited`. they will be locked in the contract indefinitely.

Although this issue is somewhat mitigated since both contracts inherit from the `MaturityVault` contract, which includes a `_mature()` function that adjusts `totalAssetDeposited` to match the current balance, this adjustment is not guaranteed to occur before withdrawals are permitted. Moreover, tokens transferred after `_mature()` has been called would still be locked.

Recommendation

It is recommended to prevent tokens from being locked in the contract.

Alleviation

[Hilter Team, 08/01/2025]: The team heeded the advice and resolved the issue in commit: 9b86f2af463028173079c7180e231c6f14325c54.

UVB-01 INCONSISTENT USE OF `Pausable` CONTRACT IN `UpsideVault`

Category	Severity	Location	Status
Access Control	Medium	vaults/UpsideVault.sol (da9877): 52, 80	Resolved

Description

The `UpsideVault` contract inherits from the `Pausable` contract, which provides functionality to pause and unpause contract operations. However, the `UpsideVault` does not utilize the `whenNotPaused` modifier from the `Pausable` contract in its `_deposit()` and `_withdraw()` functions. This oversight nullifies the benefits of having a pausable mechanism.

Recommendation

It is recommended to apply `whenNotPaused` modifier in the `UpsideVault`'s `_deposit()` and `_withdraw()` functions.

Alleviation

[Hilter Team, 08/01/2025]: The team heeded the advice and resolved the issue in commit: 9b86f2af463028173079c7180e231c6f14325c54.

HBV-01 | TOKENS TRANSFERRED TO VAULT WILL BE LOCKED

Category	Severity	Location	Status
Design Issue	Minor	base/HilterBaseVault.sol (da9877): 132, 140	Resolved

Description

The comments of the `transfer()` and `transferFrom()` functions in the `HilterBaseVault` contract indicate that tokens sent to the contract should allow for asset redemption. However, there appears to be no mechanism in the provided code to handle the tokens once they reach the contract. This could lead to tokens being locked within the contract with no clear method for users to retrieve the corresponding assets.

```
    /// @notice The share token should be soul bound. Should be transferable only to vault to receive assets back
    function transfer(address to, uint256 value) public override(ERC20, IERC20) returns (bool) {
        if (to != address(this)) revert HilterVault__TransferOutsideEcosystem();
        address owner = _msgSender();
        _transfer(owner, to, value);
        return true;
    }

    /// @notice The share token should be soul bound. Should be transferable only to vault to receive assets back
    function transferFrom(address from, address to, uint256 value) public override(ERC20, IERC20) returns (bool) {
        if (to != address(this)) revert HilterVault__TransferOutsideEcosystem();
        address spender = _msgSender();
        _spendAllowance(from, spender, value);
        _transfer(from, to, value);
        return true;
    }
```

Recommendation

It is recommended to revert any call to `transfer()` or `transferFrom()`.

Alleviation

[Hilter Team, 05/07/2025]: The team heeded the advice and resolved the issue in commit: 9b86f2af463028173079c7180e231c6f14325c54.

SRC-01 INCOMPATIBILITY WITH DEFLATIONARY TOKENS (NON-STANDARD ERC20 TOKEN)

Category	Severity	Location	Status
Volatile Code	Minor	base/HilterBaseVault.sol (da9877): 82, 106; vaults/UpsideVault.sol (da9877): 71, 72, 94, 97	Acknowledged

Description

The vault contracts are not equipped to handle non-standard ERC20 tokens, including deflationary or rebase tokens, which may apply transaction fees or adjust balances dynamically. The contract's reliance on `transferFrom()` and `transfer()` methods without validating the actual transferred amounts could result in discrepancies between expected and actual token balances. This issue is particularly critical because it could lead to insufficient tokens available for depositors when attempting to withdraw, potentially disrupting contract functionality and user transactions.

Scenario

When transferring deflationary ERC20 tokens, the input amount may not equal the received amount due to the charged transaction fee. For example, if a user sends 100 deflationary tokens (with a 10% transaction fee), only 90 tokens actually arrive to the contract. However, a failure to discount such fees may allow the same user to withdraw 100 tokens from the contract, which causes the contract to lose 10 tokens in such a transaction.

Proof of Concept

Foundry test:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity 0.7.5;
pragma abicoder v2;

import "forge-std/Test.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract DeflationaryToken is ERC20 {
    address deadAddress = 0x000000000000000000000000000000000000dEaD;
    constructor () ERC20 ("Test", "TEST") {
        _mint(msg.sender, 1000000);
    }

    function _transfer(address sender, address recipient, uint256 amount) internal
override {
        uint256 burnAmount = 10 * amount / 100;
        uint256 transferAmount = amount - burnAmount;
        super._transfer(sender, deadAddress, burnAmount);
        super._transfer(sender, recipient, transferAmount);
    }
}

contract VictimContract {
    address public token;
    mapping(address => uint256) public stakedAmount;

    constructor (address _token) {
        token = _token;
    }

    function stake(uint256 amount) public {
        ERC20(token).transferFrom(msg.sender, address(this), amount);
        stakedAmount[msg.sender] += amount;
    }

    function unstake() public {
        uint256 amount = stakedAmount[msg.sender];
        ERC20(token).transfer(msg.sender, amount);
        stakedAmount[msg.sender] = 0;
    }
}

contract DeflationaryTokenTest is Test {
    DeflationaryToken public deflationaryToken;
    VictimContract public victimContract;
    address public user = vm.addr(1);

    function setUp() public {
        deflationaryToken = new DeflationaryToken();
```

```
        victimContract = new VictimContract(address(deflationaryToken));
        deflationaryToken.transfer(address(victimContract), 10000);
        deflationaryToken.transfer(user, 10000);
    }

    function testIssue() public {
        vm.startPrank(user);
        deflationaryToken.approve(address(victimContract), 1000);

        uint256 victimBalanceBefore =
deflationaryToken.balanceOf(address(victimContract)); // balance before staking
        victimContract.stake(1000);
        victimContract.unstake();

        uint256 victimBalanceAfter =
deflationaryToken.balanceOf(address(victimContract)); // balance after unstaking

        vm.stopPrank();

        require(victimBalanceAfter < victimBalanceBefore, "error"); // victim
contract lost tokens
    }
}
```

Recommendation

We advise the client to regulate the set of tokens supported and add necessary mitigation mechanisms to keep track of accurate balances if there is a need to support non-standard ERC20 tokens.

Alleviation

[Hilter Team, 08/01/2025]: Acknowledged, our Vaults will be using well-known ERC-20 stablecoins only, e.g. USDC and USDT.

SRC-03 | THIRD-PARTY DEPENDENCIES

Category	Severity	Location	Status
Volatile Code	Minor	base/HilterBaseVault.sol (da9877): 55; vaults/UpsideVault.sol (d a 9877): 16	Acknowledged

Description

The contract is serving as the underlying entity to interact with third-parties asset token and collateral token. The scope of the audit treats third-party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets.

Recommendation

We recommend that the project team constantly monitor the functionality of these contracts to mitigate any side effects that may occur when unexpected changes are introduced.

Alleviation

[Hilter Team, 05/07/2025]:

Understood and agreed regarding third-party dependencies overall. For more context:

- Asset Token will always be a USDC or USDT stablecoin.
- Hilter will develop this contract from OpenZeppelin standards.

UVB-03 | SHADOWING STATE VARIABLE

Category	Severity	Location	Status
Coding Style	Informational	vaults/UpsideVault.sol (da9877): 23	Resolved

Description

The state variable `_balances` in the `UpsideVault` contract is shadowing the same named component in the parent contract `ERC20`. This means that when the derived contract accesses the state variable by its name, it will use the one defined in the derived contract, not the one in the parent contract. This can lead to confusion.

Recommendation

It is suggested to rename the state variable that shadows another definition.

Alleviation

[Hilter Team, 05/07/2025]: The team heeded the advice and resolved the issue in commit: 9b86f2af463028173079c7180e231c6f14325c54.

FORMAL VERIFICATION | HILTER

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

Considered Functions And Scope

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

Verification of Standard Ownable Properties

We verified *partial* properties of the public interfaces of those token contracts that implement the Ownable interface. This involves:

- function `owner` that returns the current owner,
- functions `renounceOwnership` that removes ownership,
- function `transferOwnership` that transfers the ownership to a new owner.

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

Property Name	Title
ownable-renounce-ownership-is-permanent	Once Renounced, Ownership Cannot be Regained
ownable-transferownership-correct	Ownership is Transferred.
ownable-renounceownership-correct	Ownership is Removed.
ownable-owner-succeed-normal	`owner` Always Succeeds

Verification Results

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

Detailed Results For Contract HilterKYCProvider (packages/contracts/src/HilterKYCProvider.sol) In Commit da9877c0823663a5fca5f75f3a09b84e050dab8e

Verification of Standard Ownable Properties

Detailed Results for Function `renounceOwnership`

Property Name	Final Result	Remarks
ownable-renounce-ownership-is-permanent	True	
ownable-renounceownership-correct	True	

Detailed Results for Function `transferOwnership`

Property Name	Final Result	Remarks
ownable-transferownership-correct	True	

Detailed Results for Function `owner`

Property Name	Final Result	Remarks
ownable-owner-succeed-normal	True	

APPENDIX | HILTER

Finding Categories

Categories	Description
Coding Style	Coding Style findings may not affect code behavior, but indicate areas where coding practices can be improved to make the code more understandable and maintainable.
Access Control	Access Control findings are about security vulnerabilities that make protected assets unsafe.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.

Checksum Calculation Method

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

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

Details on Formal Verification

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

The following assumptions and simplifications apply to our model:

- Certain low-level calls and inline assembly are not supported and may lead to a contract not being formally verified.
- We model the semantics of the Solidity source code and not the semantics of the EVM bytecode in a compiled contract.

Formalism for property specifications

All properties are expressed in a behavioral interface specification language that CertiK has developed for Solidity, which allows us to specify the behavior of each function in terms of the contract state and its parameters and return values, as well as contract properties that are maintained by every observable state transition. Observable state transitions occur when the contract's external interface is invoked and the invocation does not revert, and when the contract's Ether balance is changed by the EVM due to another contract's "self-destruct" invocation. The specification language has the usual Boolean connectives, as well as the operator `\old` (used to denote the state of a variable before a state transition), and several types of specification clause:

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

- `requires [cond]` - the condition `cond`, which refers to a function's parameters, return values, and contract state variables, must hold when a function is invoked in order for it to exhibit a specified behavior.
- `ensures [cond]` - the condition `cond`, which refers to a function's parameters, return values, and both `\old` and current contract state variables, is guaranteed to hold when a function returns if the corresponding requires condition held when it was invoked.
- `invariant [cond]` - the condition `cond`, which refers only to contract state variables, is guaranteed to hold at every observable contract state.
- `constraint [cond]` - the condition `cond`, which refers to both `\old` and current contract state variables, is guaranteed to hold at every observable contract state except for the initial state after construction (because there is no previous state); constraints are used to restrict how contract state can change over time.

Description of the Analyzed Ownable Properties

Properties related to function `renounceOwnership`

ownable-renounce-ownership-is-permanent

The contract must prohibit regaining of ownership once it has been renounced.

Specification:

```
constraint \old(owner()) == address(0) ==> owner() == address(0);
```

ownable-renounceownership-correct

Invocations of `renounceOwnership()` must set ownership to address(0).

Specification:

```
ensures this.owner() == address(0);
```

Properties related to function `transferOwnership`

ownable-transferownership-correct

Invocations of `transferOwnership(newOwner)` must transfer the ownership to the `newOwner`.

Specification:

```
ensures this.owner() == newOwner;
```

Properties related to function `owner`

ownable-owner-succeed-normal

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

Specification:

```
reverts_only_when false;
```

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

CertiK | Securing the Web3 World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.