CERTIK

Preliminary Comments

Beefy Smart Contract

May 11th, 2021

Summary

This report has been prepared for **Beefy Smart Contract**, 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 Manual Review and Static Analysis 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.

There are a few injection dependent external contracts invoked in the current project contracts:

- `autostrat`, `vault`, `autofarm`, `Auto`, `bifi`, `masterchef`, `want`, `wbnb` and `unirouter` in the contract **StrategyAutoCake**;
- `autostrat`, `vault`, `autofarm`, `Auto`, `bifi`, `want`, `wbnb` and `unirouter` in the contract **StrategyAutoVenus**;
- `token` and `strategy` in the contract **BeefyBurningVault**;
- `token` and `strategy` in the contract **BeefyVaultV3**.

We assume all the imported libraries/contracts in the current project are valid and non-vulnerable actors, and implementing proper logic in the current project.

There are a few owner/admin-only access functions that could update important contract states and parameters, thus introducing centralization risks. We assume the project would update the contract and call the functions with valid and proper parameters. Meanwhile, to improve the trustworthiness of the project, any dynamic runtime update in the project should be notified to the community. We recommend any plan to invoke those functions should be also considered to move to the execution queue of the Timelock contract.

Overview

Project Summary

Project Name	Beefy Smart Contract
Platform	BSC
Language	Solidity
Codebase	https://github.com/beefyfinance/beefy-certik/tree/second-batch/contracts/BIFI
Commits	https://github.com/beefyfinance/beefy-certik/commit/413faccf876cb9b9da1b9a9231d582f121c1d5a7#diff-4a15db66c263314f3be1a78305a80d62a13c1d5941e2f0d4fe86094e9107b461

Audit Summary

Delivery Date	May 11, 2021
Audit Methodology	Manual Review, Static Analysis
Key Components	strategies, vaults

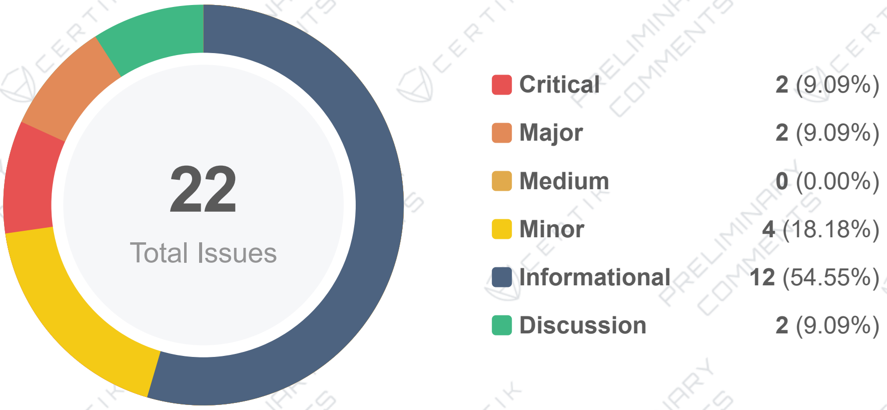
Vulnerability Summary

Total Issues	22
● Critical	2
● Major	2
● Medium	0
● Minor	4
● Informational	12
● Discussion	2

Audit Scope

ID	file	SHA256 Checksum
SAC	BIFI/strategies/Auto/StrategyAutoCake.sol	b0d6aaed418d00ee9fe955e0c3c5ea345dbf0cac84c271bb4283657d0bf9382e
SAV	BIFI/strategies/Auto/StrategyAutoVenus.sol	ece78a541758d6f6f2073a1c57d5be059d98785ee6aa6dcf6c15bc2e3f1e5c34
BBV	BIFI/vaults/BeefyBurningVault.sol	a08406f927b70d41096737e70117c9579fd360ce5aa93c09a0d7621d83454f54
BVV	BIFI/vaults/BeefyVaultV3.sol	1f3f4ebf4cf02282bb8fde32df81edf339598833413d7c0c9fcb064e604ee0b8

Findings



ID	Title	Category	Severity	Status
BBV-01	Keyword Incompatible With Solidity Version	Language Specific	● Informational	Pending
BBV-02	Function Should Be Declared External	Gas Optimization	● Informational	Pending
BBV-03	Centralization Risks	Logical Issue	● Major	Pending
BBV-04	Lack of Check for Reentrancy	Logical Issue	● Minor	Pending
BVV-01	Function Should Be Declared External	Gas Optimization	● Informational	Pending
BVV-02	Keyword Incompatible With Solidity Version	Language Specific	● Informational	Pending
BVV-03	Centralization Risks	Logical Issue	● Major	Pending
BVV-04	Lack of Check for Reentrancy	Logical Issue	● Minor	Pending
SAC-01	Lack of Return Value Handling	Logical Issue	● Informational	Pending
SAC-02	Non-Optimal Param Set	Logical Issue	● Informational	Pending
SAC-03	Lack of Check for Reentrancy	Logical Issue	● Minor	Pending
SAC-04	Function Should Be Declared External	Gas Optimization	● Informational	Pending
SAC-05	Vulnerable Contract Check for `msg.sender`	Logical Issue	● Critical	Pending
SAC-06	Repeated Function Call `farm`	Gas Optimization	● Discussion	Pending
SAC-07	Centralization Risks	**Centralization / Privilege**	● **Informational**	**Pending**
SAV-01	Lack of Return Value Handling	Logical Issue	● Informational	Pending

ID	Title	Category	Severity	Status
SAV-02	Non-Optimal Param Set	Logical Issue	● Informational	(!) Pending
SAV-03	Lack of Check for Reentrancy	Logical Issue	● Minor	(!) Pending
SAV-04	Function Should Be Declared External	Gas Optimization	● Informational	(!) Pending
SAV-05	Vulnerable Contract Check for `msg.sender`	Logical Issue	● Critical	(!) Pending
SAV-06	Mismatch Between Comment and Code	Logical Issue	● Discussion	(!) Pending
SAV-07	Centralization Risks	**Centralization / Privilege**	● **Informational**	(!) **Pending**

BBV-01 | Keyword Incompatible With Solidity Version

Category	Severity	Location	Status
Language Specific	● Informational	BIFI/vaults/BeefyBurningVault.sol: 3, 37	Pending

Description

The keyword `immutable` declared in L37, is introduced in Solidity v0.6.5.

```
37  uint256 public immutable approvalDelay;
```

However, the current contract applies Solidity v0.6.0 which does not support `immutable`:

```
 3  pragma solidity ^0.6.0;
```

Recommendation

We recommend the team review the code and apply a proper Solidity version.

BBV-02 | Function Should Be Declared External

Category	Severity	Location	Status
Gas Optimization	● Informational	BIFI/vaults/BeefyBurningVault.sol: 91, 167, 182	ⓘ Pending

Description

The functions which are never called internally within the contract should have external visibility. For example, `getPricePerFullShare`, `proposeStrat`, and `upgradeStrat`.

Recommendation

We recommend modifying the visibility of the aforementioned functions to `external`.

BBV-03 | Centralization Risks

Category	Severity	Location	Status
Logical Issue	● Major	BIFI/vaults/BeefyBurningVault.sol: 167, 182	ⓘ Pending

Description

The function `proposeStrat` in L167 allows the owner to modify the state variable `stratCandidate`, and thus updating `stratCandidate` to a new candidate strategy:

```
function proposeStrat(address _implementation) public onlyOwner {
    stratCandidate = StratCandidate({
        implementation: _implementation,
        proposedTime: block.timestamp
     });
    ...
}
```

Meanwhile, the function `upgradeStrat` in L182 replaces the active strategy with the candidate strategy updated in the function `proposeStrat` above:

```
function upgradeStrat() public onlyOwner {
    ...
    IBurningStrategy(strategy).retireStrat();
    strategy = stratCandidate.implementation;
    ...
    earn();
}
```

Our concern is, if the owner accidentally and improperly calls the function `proposeStrat` and updates the candidate strategy to a vulnerable one, and then calls the function `upgradeStrat` to apply the new candidate strategy, it might cause some unexpected loss.

Recommendation

We recommend the team review the design and ensure minimum centralization risk. Meanwhile, we recommend any plan to invoke those functions should be considered to move to the execution queue of the Timelock contract, and any dynamic runtime update in the project should be notified to the community in advance.

BBV-04 | Lack of Check for Reentrancy

Category	Severity	Location	Status
Logical Issue	● Minor	BIFI/vaults/BeefyBurningVault.sol: 145~161, 106~121	ⓘ Pending

Description

In the function `deposit` in L106, there is a state update after an external call:

```
    function deposit(uint _amount) public {
        ...
        token.safeTransferFrom(msg.sender, address(this), _amount);    // external call
        ...
        _mint(msg.sender, shares);    // chain state update
    }
```

Similarly, in the function `withdraw` in L145, there is a state update after an external call:

```
    function withdraw(uint256 _shares) public {
        ...
        IBurningStrategy(strategy).withdraw(_withdraw);    // external call
        ...
        token.safeTransfer(msg.sender, r);    // chain state update
    }
```

In these cases, reentrancy guard rail is highly recommended to prevent reentrancy attack.

Recommendation

We recommend applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned function to prevent reentrancy attack.

BVV-01 | Function Should Be Declared External

Category	Severity	Location	Status
Gas Optimization	● Informational	BIFI/vaults/BeefyVaultV3.sol: 89, 165, 180	ⓘ Pending

Description

The functions which are never called internally within the contract should have external visibility. For example, `getPricePerFullShare`, `proposeStrat` and `upgradeStrat`.

Recommendation

We recommend modifying the visibility of the aforementioned functions to `external`.

BVV-02 | Keyword Incompatible With Solidity Version

Category	Severity	Location	Status
Language Specific	● Informational	BIFI/vaults/BeefyVaultV3.sol: 35	Pending

Description

The keyword `immutable` declared in L35, is introduced in Solidity v0.6.5.

```
35  uint256 public immutable approvalDelay;
```

However, the current contract applies Solidity v0.6.0 which does not support `immutable`:

```
3  pragma solidity ^0.6.0;
```

Recommendation

We recommend the team review the code and apply a proper Solidity version.

BVV-03 | Centralization Risks

Category	Severity	Location	Status
Logical Issue	● Major	BIFI/vaults/BeefyVaultV3.sol: 165, 180	ⓘ Pending

Description

The function `proposeStrat` in L165 allows the owner to modify the state variable `stratCandidate`, and thus updating `stratCandidate` to a new candidate strategy:

```
function proposeStrat(address _implementation) public onlyOwner {
    stratCandidate = StratCandidate({
        implementation: _implementation,
        proposedTime: block.timestamp
     });
    ...
}
```

Meanwhile, the function `upgradeStrat` in L180 replaces the active strategy with the new candidate strategy updated in the function `proposeStrat` above:

```
function upgradeStrat() public onlyOwner {
    ...
    IStrategy(strategy).retireStrat();
    strategy = stratCandidate.implementation;
    ...
    earn();
}
```

Our concern is, if the owner accidentally and improperly calls the function `proposeStrat` and updates the candidate strategy to a vulnerable one, and then calls the function `upgradeStrat` to apply the new candidate strategy, it might lead to some unexpected loss.

Recommendation

We recommend the team review the design and ensure minimum centralization risk. Meanwhile, we recommend any plan to invoke those functions should be considered to move to the execution queue of the Timelock contract, and any dynamic runtime update in the project should be notified to the community in advance.

BVV-04 | Lack of Check for Reentrancy

Category	Severity	Location	Status
Logical Issue	● Minor	BIFI/vaults/BeefyVaultV3.sol: 104~119, 143~159	ⓘ Pending

Description

In the function `deposit` in L104, there is a state update after an external call:

```
function deposit(uint _amount) public {
    ...
    token.safeTransferFrom(msg.sender, address(this), _amount);    // external call
    ...
    _mint(msg.sender, shares);    // chain state update
}
```

Similarly, in the function `withdraw` in L143, there is a state update after an external call:

```
function withdraw(uint256 _shares) public {
    ...
    IStrategy(strategy).withdraw(_withdraw);    // external call
    ...
    token.safeTransfer(msg.sender, r);    // chain state update
}
```

In these cases, reentrancy guard rail is highly recommended to prevent reentrancy attack.

Recommendation

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

SAC-01 | Lack of Return Value Handling

Category	Severity	Location	Status
Logical Issue	● Informational	BIFI/strategies/Auto/StrategyAutoCake.sol: 181, 190, 204, 255	Pending

Description

The functions `swapExactTokensForTokens` and `transfer` are not void-returning functions per IUniswapV2Router02 and IERC20 interfaces. In the `StrategyAutoCake` contract, return values of the functions are not handled properly. For instance,

```
181  IUniswapRouter(unirouter).swapExactTokensForTokens(toWbnb, 0, autoToWbnbRoute, address(this), now.add(600));
```

and

```
255  IERC20(want).transfer(vault, wantBal);
```

Ignoring the return values of these functions might cause some unexpected exceptions, especially if the called functions don't revert automatically when failing.

Recommendation

We recommend checking the output of the aforementioned functions before continuing processing.

SAC-02 | Non-Optimal Param Set

Category	Severity	Location	Status
Logical Issue	● Informational	BIFI/strategies/Auto/StrategyAutoCake.sol: 181, 190, 204	ⓘ Pending

Description

According to the official document of Uniswap, the 2nd input parameter, `amountOutMin` of the function `swapExactTokensForTokens` indicates the desired minimum amount of tokens that should be swapped. If less than `amountOutMin` is swapped, this function will revert. However, in this contract the parameter `amountOutMin` is set as 0, for instance,

```
181  IUniswapRouter(unirouter).swapExactTokensForTokens(toWbnb, 0, autoToWbnbRoute,
address(this), now.add(600));
```

This will result in an instant token-swap without considering the market price. Therefore, it is vulnerable to front running attack or sandwich attack.

Recommendation

We recommend carefully setting up the parameter `amountOutMin` in aforementioned lines as some meaningful non-zero values, to reduce the potential risks.

SAC-03 | Lack of Check for Reentrancy

Category	Severity	Location	Status
Logical Issue	● Minor	BIFI/strategies/Auto/StrategyAutoCake.sol: 162	Pending

Description

In the function `harvest`, there are state updates (within `chargeFees`, `swapRewards` and `deposit`) and event emit (`StratHarvest`) after external call `IAutoFarmV2.deposit`, and thus is vulnerable to reentrancy attack.

```
    function harvest() external whenNotPaused {
        require(!Address.isContract(msg.sender), "!contract");
        IAutoFarmV2(autofarm).deposit(poolId, 0);
        chargeFees();
        swapRewards();
        deposit();

        emit StratHarvest(msg.sender);
    }
```

Recommendation

We recommend applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned function to prevent reentrancy attack.

SAC-04 | Function Should Be Declared External

Category	Severity	Location	Status
Gas Optimization	● Informational	BIFI/strategies/Auto/StrategyAutoCake.sol: 211, 261	ⓘ Pending

Description

The functions that are never called internally within the contract should have external visibility. For example, `balanceOf` and `panic`.

Recommendation

We recommend changing the visibility of the aforementioned functions to `external`.

SAC-05 | Vulnerable Contract Check for `msg.sender`

Category	Severity	Location	Status
Logical Issue	● Critical	BIFI/strategies/Auto/StrategyAutoCake.sol: 163	ⓘ Pending

Description

The `require` statement in L163 in the function `harvest` is supposed to prevent the function from being called by a contract:

```
require(!Address.isContract(msg.sender), "!contract");
```

The function `isContract` in L163 is from the library `@openzeppelin/contracts/utils/Address.sol`, and it will return `false` if `extcodesize` returns 0:

```
function isContract(address account) internal view returns (bool) {
    ...
    uint256 size;
    assembly { size := extcodesize(account) }
    return size > 0;
}
```

However, `Address.isContract(msg.sender)==false` cannot 100% guarantee the caller is a non-contract user. When the function `harvest` is called from the constructor of another contract, `extcodesize` returns 0 and the function `isContract` will return `false`. In this case the `require` check in L163 will pass instead of reverting.

Recommendation

We recommend checking by `msg.sender == tx.origin` to exclude function calls invoked by other contracts.

SAC-06 | Repeated Function Call `farm`

Category	Severity	Location	Status
Gas Optimization	● Discussion	BIFI/strategies/Auto/StrategyAutoCake.sol: 139, 140	ⓘ Pending

Description

In the function `withdraw`, `IStratX(autostrat).farm()` is executed twice:

```
function withdraw(uint256 _amount) external {
    ...
    IStratX(autostrat).farm();
    IStratX(autostrat).farm();
    ...
}
```

We are curious if this is intended and if it is necessary to `farm` twice.

SAC-07 | Centralization Risks

Category	Severity	Location	Status
Centralization / Privilege	● **Informational**	BIFI/strategies/Auto/StrategyAutoCake.sol: 249~256	ⓘ **Pending**

Description

In L249, the function `retireStrat` allows `vault` to retire an AutoFarm strategy by withdrawing all the tokens from `autofarm` and then transferring the tokens to `vault`:

```
249     function retireStrat() external {
250         require(msg.sender == vault, "!vault");
251 
252         IAutoFarmV2(autofarm).emergencyWithdraw(poolId);
253 
254         uint256 wantBal = IERC20(want).balanceOf(address(this));
255         IERC20(want).transfer(vault, wantBal);
256     }
```

Our concern is, if the function is accidentally called by the `vault`, the project/users might suffer from unexpected loss.

Recommendation

We recommend the team confirm the `vault` of the contract is set up correctly and `retireStrat` is only called in emergency. Meanwhile, any plan to invoke the function `retireStrat` should be considered to move to an execution queue of the Timelock contract. Any dynamic runtime update in the project should be notified to the community in advance.

SAV-01 | Lack of Return Value Handling

Category	Severity	Location	Status
Logical Issue	● Informational	BIFI/strategies/Auto/StrategyAutoVenus.sol: 175, 184, 198, 241	ⓘ Pending

Description

The functions `swapExactTokensForTokens` and `transfer` are not void-returning functions per IUniswapV2Router02 and IERC20 interfaces. In this contract, return values of the functions are not handled properly. For instance,

```
175  IUniswapRouter(unirouter).swapExactTokensForTokens(toWbnb, 0, autoToWbnbRoute, address(this), now.add(600));
```

and

```
241  IERC20(want).transfer(vault, wantBal);
```

Ignoring the return values of these functions might cause some unexpected exceptions, especially if the called functions don’t revert automatically when failing.

Recommendation

We recommend checking the output of the aforementioned functions before continuing processing.

SAV-02 | Non-Optimal Param Set

Category	Severity	Location	Status
Logical Issue	● Informational	BIFI/strategies/Auto/StrategyAutoVenus.sol: 175, 184, 198	ⓘ Pending

Description

According to the official document of Uniswap, the 2nd input parameter, `amountOutMin` of the function `swapExactTokensForTokens` indicates the desired minimum amount of tokens that should be swapped. If less than `amountOutMin` is swapped, this function will revert. However, in this contract the parameter `amountOutMin` is always set as 0, for instance,

```
 175  IUniswapRouter(unirouter).swapExactTokensForTokens(toWbnb, 0, autoToWbnbRoute,
address(this), now.add(600));
```

This will result in an instant token-swap without considering the market price. Therefore, it is vulnerable to front running attack or sandwich attack.

Recommendation

We recommend carefully setting up the `amountOutMin` parameter in aforementioned lines as some meaningful non-zero values, to reduce the potential risks.

SAV-03 | Lack of Check for Reentrancy

Category	Severity	Location	Status
Logical Issue	● Minor	BIFI/strategies/Auto/StrategyAutoVenus.sol: 156	Pending

Description

In the function `harvest` there are state updates (within `chargeFees`, `swapRewards` and `deposit`) and event emit (`StratHarvest`) after the external call `IAutoFarmV2.deposit`, and thus is vulnerable to reentrancy attack.

```
156     function harvest() external whenNotPaused {
157         require(!Address.isContract(msg.sender), "!contract");
158         IAutoFarmV2(autofarm).deposit(poolId, 0);
159         chargeFees();
160         swapRewards();
161         deposit();
162 
163         emit StratHarvest(msg.sender);
164     }
```

Recommendation

We recommend applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned function to prevent reentrancy attack.

SAV-04 | Function Should Be Declared External

Category	Severity	Location	Status
Gas Optimization	● Informational	BIFI/strategies/Auto/StrategyAutoVenus.sol: 213, 247	ⓘ Pending

Description

The functions that are never called internally within the contract should have external visibility. For example, `balanceOf` and `panic`.

Recommendation

We recommend changing the visibility of the aforementioned functions to `external`.

SAV-05 | Vulnerable Contract Check for `msg.sender`

Category	Severity	Location	Status
Logical Issue	● Critical	BIFI/strategies/Auto/StrategyAutoVenus.sol: 157	ⓘ Pending

Description

The `require` check in L157 in the function `harvest` is supposed to prevent the function from being called by a contract:

```
require(!Address.isContract(msg.sender), "!contract");
```

The function `isContract` in L157 is from the library `@openzeppelin/contracts/utils/Address.sol`, and it will return `false` if `extcodesize` returns 0:

```
function isContract(address account) internal view returns (bool) {
    ...
    uint256 size;
    assembly { size := extcodesize(account) }
    return size > 0;
}
```

However, `Address.isContract(msg.sender)==false` cannot 100% guarantee the caller is a non-contract user. When the function `harvest` is called from the constructor of another contract, `extcodesize` returns 0 and the function `isContract` will return `false`. In this case the `require` check in L157 will pass instead of reverting.

Recommendation

We recommend checking by `msg.sender == tx.origin` to exclude function calls invoked by other contracts.

SAV-06 | Mismatch Between Comment and Code

Category	Severity	Location	Status
Logical Issue	● Discussion	BIFI/strategies/Auto/StrategyAutoVenus.sol: 132, 136~154	ⓘ Pending

Description

According to the comment in L132, the function `withdraw` is supposed to call the `farm` function of the strategy `autostrat` before withdrawing the tokens from the strategy, in case there are some omissions left in the strategy:

```
     * It redeposits harvested and pending cakes in AutoFarm strategy via farm()
```

From the code implementation, however, the function `farm` is not executed within the function `withdraw`.

Recommendation

We recommend the team check the implementation logic and confirm if `farm` of `autostrat` should be executed first in the function `withdraw`.

SAV-07 | Centralization Risks

Category	Severity	Location	Status
Centralization / Privilege	● **Informational**	BIFI/strategies/Auto/StrategyAutoVenus.sol: 235~242	ⓘ **Pending**

Description

In L235, the function `retireStrat` allows `vault` to retire an AutoFarm strategy by withdrawing all the tokens from `autofarm` and then transferring the tokens to `vault`:

```
235     function retireStrat() external {
236         require(msg.sender == vault, "!vault");
237 
238         IAutoFarmV2(autofarm).emergencyWithdraw(poolId);
239 
240         uint256 wantBal = IERC20(want).balanceOf(address(this));
241         IERC20(want).transfer(vault, wantBal);
242     }
```

Our concern is, if the function is accidentally called by the `vault`, the project/users might suffer from unexpected loss.

Recommendation

We recommend the team confirm the `vault` of the contract is set up correctly and `retireStrat` is only called in emergency. Meanwhile, any plan to invoke the function `retireStrat` should be also considered to move to an execution queue of a Timelock contract and any dynamic runtime update in the project should be notified to the community in advance.

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.

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 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.

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.

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 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.