CERTIK

Security Assessment

Beefy Smart Contract

Jun 24th, 2021

Table of Contents

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
Commit	https://github.com/beefyfinance/beefy-certik/tree/413faccf876cb9b9da1b9a9231d582f121c1d5a7 https://github.com/beefyfinance/beefy-certik/tree/617b48ec12a84f03b77f21f08762deb2fe8a2d2d

Audit Summary

Delivery Date	Jun 24, 2021
Audit Methodology	Manual Review, Static Analysis
Key Components	strategies, vaults

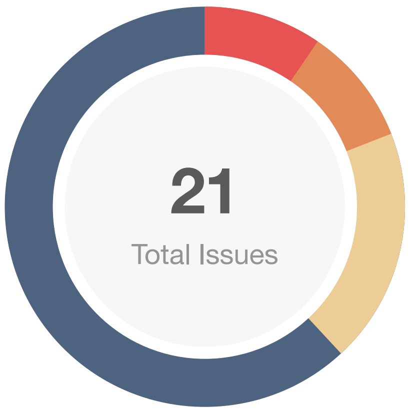
Vulnerability Summary

Vulnerability Level	Total	Pending	Partially Resolved	Resolved	Acknowledged	Declined
● Critical	2	0	0	2	0	0
● Major	2	0	0	0	2	0
● Medium	0	0	0	0	0	0
● Minor	4	0	1	2	1	0
● Informational	13	0	2	7	4	0
● Discussion	0	0	0	0	0	0

Audit Scope

ID	file	SHA256 Checksum
SAC	BIFI/strategies/Auto/StrategyAutoCake.sol	ab19930203eafdd0c9531f9518d9a07da8fc6acd7d125b42ac4c43672d7bbab5
SAV	BIFI/strategies/Auto/StrategyAutoVenus.sol	ec1b0506ebe9a6996ec5cd3f95a32cbf55b09815f64767c58db8c7fb4bf1e1a6
BBV	BIFI/vaults/BeefyBurningVault.sol	bb24dc6e642c5f24cbb03fa81e5c34f82838af4191cd72262c1931cf6f61391c
BVV	BIFI/vaults/BeefyVaultV3.sol	7b319f87fe8c37d766a25ef0d29c08cfe1b9d062d4356026eb602bba3baf3816

Findings



🟥 **Critical**	**2** (9.52%)
🟧 **Major**	**2** (9.52%)
🟨 **Medium**	**0** (0.00%)
🟨 **Minor**	**4** (19.05%)
🟦 **Informational**	**13** (61.90%)
🟩 **Discussion**	**0** (0.00%)

ID	Title	Category	Severity	Status
BBV-01	Keyword Incompatible With Solidity Version	Language Specific	● Informational	✓ Resolved
BBV-02	Function Should Be Declared External	Gas Optimization	● Informational	✓ Resolved
BBV-03	Centralization Risk	Logical Issue	● Major	ⓘ Acknowledged
BBV-04	Lack of Check for Reentrancy	Logical Issue	● Minor	ⓘ Acknowledged
BVV-01	Function Should Be Declared External	Gas Optimization	● Informational	✓ Resolved
BVV-02	Keyword Incompatible With Solidity Version	Language Specific	● Informational	✓ Resolved
BVV-03	Centralization Risk	Logical Issue	● Major	ⓘ Acknowledged
BVV-04	Lack of Check for Reentrancy	Logical Issue	● Minor	Partially Resolved
SAC-01	Lack of Return Value Handling	Logical Issue	● Informational	Partially Resolved
SAC-02	Non-Optimal Param Set	Logical Issue	● Informational	ⓘ Acknowledged
SAC-03	Lack of Check for Reentrancy	Logical Issue	● Minor	✓ Resolved
SAC-04	Function Should Be Declared External	Gas Optimization	● Informational	✓ Resolved
SAC-05	Vulnerable Contract Check for `msg.sender`	Logical Issue	● Critical	✓ Resolved

ID	Title	Category	Severity	Status
SAC-07	Centralization Risk	**Centralization / Privilege**	● **Informational**	ⓘ **Acknowledged**
SAV-01	Lack of Return Value Handling	Logical Issue	● Informational	Partially Resolved
SAV-02	Non-Optimal Param Set	Logical Issue	● Informational	ⓘ Acknowledged
SAV-03	Lack of Check for Reentrancy	Logical Issue	● Minor	✓ Resolved
SAV-04	Function Should Be Declared External	Gas Optimization	● Informational	✓ Resolved
SAV-05	Vulnerable Contract Check for `msg.sender`	Logical Issue	● Critical	✓ Resolved
SAV-06	Mismatch Between Comment and Code	Logical Issue	● Informational	✓ Resolved
SAV-07	Centralization Risk	**Centralization / Privilege**	● **Informational**	ⓘ **Acknowledged**

BBV-01 | Keyword Incompatible With Solidity Version

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

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.

Alleviation

The development team heeded our advice and resolved this issue in the commit 8bebeb39703bfee88f84e6d962a2f07141b3e908.

BBV-02 | Function Should Be Declared External

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

Description

The functions which are never called internally within the contract should have external visibility. For example,

- `getPricePerFullShare()`
- `proposeStrat()`
- `upgradeStrat()`

Recommendation

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

Alleviation

The development team heeded our advice and resolved this issue in the commit d73c66df1e59fcc724dc7a32c065718c37672a87.

BBV-03 | Centralization Risk

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

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.

Alleviation

(**Beefy Team Response**)

- There is already a custom timelock in place in the code sent. It is the reason why the upgrade feature is split into two functions (proposeStrat and upgradeStrat).
- The community gets automatic notifications on all strategy upgrades, both on telegram and on discord.
- As an added security measure we are now also using an OpenZeppelin Timelock contract, owned by a multisig, to manage all the ‘ownership’ features, like handling these two functions.

BBV-04 | Lack of Check for Reentrancy

Category	Severity	Location	Status
Logical Issue	● Minor	BIFI/vaults/BeefyBurningVault.sol: 145~161	ⓘ Acknowledged

Description

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	✓ Resolved

Description

The functions which are never called internally within the contract should have external visibility. For example,

- `getPricePerFullShare`
- `proposeStrat()`
- `upgradeStrat()`

Recommendation

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

Alleviation

The development team heeded our advice and resolved this issue in the commit 4b2ca84c607bb3a5eb7bc3362aab97f5dbdd1baf.

BVV-02 | Keyword Incompatible With Solidity Version

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

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.

Alleviation

The development team heeded our advice and resolved this issue in the commit 4c90d60daadc7ef08a07979f641195f853e635d0.

BVV-03 | Centralization Risk

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

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.

Alleviation

(**Beefy Team Response**)

- There is already a custom timelock in place in the code sent. It is the reason why the upgrade feature is split into two functions (proposeStrat and upgradeStrat).
- The community gets automatic notifications on all strategy upgrades, both on telegram and on discord.
- As an added security measure we are now also using an OpenZeppelin Timelock contract, owned by a multisig, to manage all the ‘ownership’ features, like handling these two functions.

BVV-04 | Lack of Check for Reentrancy

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

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.

Alleviation

The development team heeded our advice and applied reentrancy guard rail for the function `deposit` in the commit 30285db835dd31b5aaef1765ed692c497acc8023. We recommend also applying `nonReentrant` modifier for `withdraw`.

SAC-01 | Lack of Return Value Handling

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

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.

Alleviation

The development team heeded our advice and replaced `transfer` with `safeTransfer` in the commit ba91656f4f27c415d6605d2f8ab0886bc75cbc20, and left `swapExactTokensForTokens` as it was.

(**CertiK**)

Per the current **PancakeRouter** contract design, the function `swapExactTokensForTokens` will automatically revert on failure, which is safe. However, we encourage the team to be cautious about any modification or update of **PancakeRouter** to ensure these external calls coordinate well with your project logic.

SAC-02 | Non-Optimal Param Set

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

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	✓ Resolved

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.

Alleviation

The development team heeded our advice and resolved this issue in the commit 247e4e5400260293ee06868bb02bf466d5483e4f.

SAC-04 | Function Should Be Declared External

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

Description

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

- `balanceOf()`
- `panic()`

Recommendation

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

Alleviation

The development team heeded our advice and resolved this issue in the commit eee87610f905e12a45596ae09299c48c7ad7416d.

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

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

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.

Alleviation

The development team heeded our advice and resolved this issue in the commit 136df3b398928f2cc261cf815f408b1b6f03e350.

SAC-07 | Centralization Risk

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

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.

Alleviation

(**Beefy Team Response**)

- If this accidental call of the retireStrat() function were to happen, users would still be able to withdraw their corresponding CAKE from the vault contract.
- The call from the vault to activate this retireStrat() is indeed behind a Timelock on our vaults.

SAV-01 | Lack of Return Value Handling

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

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.

Alleviation

The development team heeded our advice and replaced `transfer` with `safeTransfer` in the commit 1993ab37db6f5f1c432cc6f70372899d902000a7, and left `swapExactTokensForTokens` as it was.

(**CertiK**)

Per the current **PancakeRouter** contract design, the function `swapExactTokensForTokens` will automatically revert on failure, which is safe. However, we encourage the team to be cautious about any modification or update of **PancakeRouter** to ensure these external calls coordinate well with your project logic.

SAV-02 | Non-Optimal Param Set

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

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	✓ Resolved

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.

Alleviation

The development team heeded our advice and resolved this issue in the commit de50dc0ce342b5cbe6c88a49d71970f3371ac1fb.

SAV-04 | Function Should Be Declared External

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

Description

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

- `balanceOf()`
- `panic()`

Recommendation

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

Alleviation

The development team heeded our advice and resolved this issue in the commit 28f579b19a662f1d2978397198f0a167814c9fcc.

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

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

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.

Alleviation

The development team heeded our advice and resolved this issue in the commit 214ee5c95fc58d3271d990dc6c73392130701bd4.

SAV-06 | Mismatch Between Comment and Code

Category	Severity	Location	Status
Logical Issue	● Informational	BIFI/strategies/Auto/StrategyAutoVenus.sol: 132, 136~154	✓ Resolved

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`.

Alleviation

The development team heeded our advice and resolved this issue in the commit 617b48ec12a84f03b77f21f08762deb2fe8a2d2d.

SAV-07 | Centralization Risk

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

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`:

```
    function retireStrat() external {
        require(msg.sender == vault, "!vault");

        IAutoFarmV2(autofarm).emergencyWithdraw(poolId);

        uint256 wantBal = IERC20(want).balanceOf(address(this));
        IERC20(want).transfer(vault, wantBal);
    }
```

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.

Alleviation

(**Beefy Team Response**)

- If this accidental call of the retireStrat() function were to happen, users would still be able to withdraw their corresponding CAKE from the vault contract.
- The call from the vault to activate this retireStrat() is indeed behind a Timelock on our vaults.

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.

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.

Language Specific

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

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

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.

CERTIK