BLACKHOLE PROTOCOL

Smart Contract Review

Deliverable: Smart Contract Audit Report

Security Report

September 2021

Disclaimer

The information and views set out in this publication are those of the author(s) and do not necessarily reflect the official opinion of the Company. The content, conclusions and recommendations set out in this publication are elaborated in the specific for only project.

eNebula Solutions does not guarantee the authenticity of the project or organization or team of members that is connected/owner behind the project or nor accuracy of the data included in this study. All representations, warranties, undertakings and guarantees relating to the report are excluded, particularly concerning – but not limited to – the qualities of the assessed projects and products. Neither the Company nor any personating on the Company's behalf may be held responsible for the use that may be made of the information contained herein.

eNebula Solutions retains the right to display audit reports and other content elements as examples of their work in their portfolio and as content features in other projects with protecting all security purpose of customer. The report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities fixed - upon a decision of the Customer.

Report Summary

Title	**BLACKHOLE PROTOCOL Smart Contract Audit**		
Project Owner	BLACKHOLE PROTOCOL		
Type	Public		
Reviewed by	Vatsal Raychura	**Revision date**	07/09/2021
Approved by	eNebula Solutions Private Limited	**Approval date**	07/09/2021
		Nº Pages	**45**

Overview

Background

BLACKHOLE PROTOCOL requested that eNebula Solutions perform an Extensive Smart Contract audit of their Smart Contract.

Project Dates

The following is the project schedule for this review and report:

- **September 05**: Smart Contract Review Completed *(Completed)*
- **September 05**: Delivery of Smart Contract Audit Report *(Completed)*
- **September 07**: Delivery of Smart Contract Re-Audit Report *(Completed)*

Review Team

The following eNebula Solutions team member participated in this review:

- Sejal Barad, Security Researcher and Engineer
- Vatsal Raychura, Security Researcher and Engineer

Coverage

Target Specification and Revision

For this audit, we performed research, investigation, and review of the smart contract of BLACKHOLE PROTOCOL.

The following documentation repositories were considered in-scope for the review:

- BLACKHOLE PROTOCOL Project: https://bscscan.com/address/0x112Dac520F63fa8Bf6f4B9e922FaccE5E1b320E2#code

Introduction

Given the opportunity to review BLACKHOLE PROTOCOL Project's smart contract source code, we in the report outline our systematic approach to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistencies between smart contract code and design document, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts is ready to launch after resolving the mentioned issues, there are no critical or high issues found related to business logic, security or performance.

About BLACKHOLE PROTOCOL : -

Item	Description
Issuer	BLACKHOLE PROTOCOL
Website	https://blackhole.black/#/
Type	BEP20
Platform	Solidity
Audit Method	Whitebox
Latest Audit Report	September 07, 2021

The Test Method Information: -

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open-source code, non-open-source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant effect on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project party should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.

The Full List of Check Items:

Category	Check Item
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	MONEY-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead of Transfer
	Costly Loop
	(Unsafe) Use of Untrusted Libraries
	(Unsafe) Use of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses
Semantic Consistency Checks	Semantic Consistency Checks
	Business Logics Review

Advanced DeFi Scrutiny	Functionality Checks
	Authentication Management
	Access Control & Authorization
	Oracle Security
	Digital Asset Escrow
	Kill-Switch Mechanism
	Operation Trails & Event Generation
	ERC20 Idiosyncrasies Handling
	Frontend-Contract Integration
	Deployment Consistency
	Holistic Risk Management
Additional Recommendations	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

Common Weakness Enumeration (CWE) Classifications Used in This Audit:

Category	Summary
Configuration	Weaknesses in this category are typically introduced during the configuration of the software.
Data Processing Issues	Weaknesses in this category are typically found in functionality that processes data.
Numeric Errors	Weaknesses in this category are related to improper calculation or conversion of numbers.
Security Features	Weaknesses in this category are concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management. (Software security is not security software.)
Time and State	Weaknesses in this category are related to the improper management of time and state in an environment that supports simultaneous or near-simultaneous computation by multiple systems, processes, or threads.
Error Conditions, Return Values, Status Codes	Weaknesses in this category include weaknesses that occur if a function does not generate the correct return/status code, or if the application does not handle all possible return/status codes that could be generated by a function.
Resource Management	Weaknesses in this category are related to improper management of system resources.

Behavioral Issues	Weaknesses in this category are related to unexpected behaviors from code that an application uses.
Business Logics	Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application.
Initialization and Cleanup	Weaknesses in this category occur in behaviors that are used for initialization and breakdown.
Arguments and Parameters	Weaknesses in this category are related to improper use arguments or parameters within function calls.
Expression Issues	Weaknesses in this category are related to incorrectly written expressions within code.
Coding Practices	Weaknesses in this category are related to coding practices that are deemed unsafe and increase the chances that an ex pilotable vulnerability will be present in the application. They may not directly introduce a vulnerability, but indicate the product has not been carefully developed or maintained.

Findings

Summary

Here is a summary of our findings after analyzing the BLACKHOLE PROTOCOL's Smart Contract. During the first phase of our audit, we studied the smart contract sourcecode and ran our in-house static code analyzer through the Specific tool. The purpose here is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by tool. We further manually review businesslogics, examine system operations, and place DeFi-related aspects under scrutinyto uncover possible pitfalls and/or bugs.

Severity	No. of Open Issues	No. of Closed Issues
Critical	0	0
High	0	2(Resolved/Acknowledged)
Medium	0	1(Resolved/Acknowledged)
Low	0	17(Resolved/Acknowledged)
Total	0	20

We have so far identified that there are potential issues with severity of **0 Critical, 2 High, 1 Medium, and 17 Low**. Overall, these smart contracts are well- designed and engineered.

Functional Overview

($) = payable function # = non-constant function	[Pub] public [Ext] external [Prv] private [Int] internal

+ [Lib] Math
  - [Int] max
  - [Int] min
  - [Int] average

+ [Lib] SafeMath
  - [Int] add
  - [Int] sub
  - [Int] sub
  - [Int] sub0
  - [Int] mul
  - [Int] div
  - [Int] div
  - [Int] mod
  - [Int] mod

+ [Lib] Address
  - [Int] isContract
  - [Int] sendValue #

+ [Int] IERC20
  - [Ext] totalSupply

- [Ext] balanceOf
- [Ext] transfer #
- [Ext] allowance
- [Ext] approve #
- [Ext] transferFrom #

+ [Lib] SafeERC20
- [Int] safeTransfer #
- [Int] safeTransferFrom #
- [Int] safeApprove #
- [Int] safeIncreaseAllowance #
- [Int] safeDecreaseAllowance #
- [Prv] _callOptionalReturn #

+ Proxy
- [Ext] <Fallback> ($)
- [Ext] <Fallback> ($)
- [Int] _implementation
- [Int] _delegate #
- [Int] _willFallback #
- [Int] _fallback #

+ BaseUpgradeabilityProxy (Proxy)
- [Int] _implementation
- [Int] _upgradeTo #
- [Int] _setImplementation #

+ BaseAdminUpgradeabilityProxy (BaseUpgradeabilityProxy)
- [Ext] admin #
  - modifiers: ifAdmin
- [Ext] implementation #

```
    - modifiers: ifAdmin
  - [Ext] changeAdmin #
    - modifiers: ifAdmin
  - [Ext] upgradeTo #
    - modifiers: ifAdmin
  - [Ext] upgradeToAndCall ($)
    - modifiers: ifAdmin
  - [Int] _admin
  - [Int] _setAdmin #
  - [Int] _willFallback #

+ [Int] IAdminUpgradeabilityProxyView
  - [Ext] admin
  - [Ext] implementation

+  UpgradeabilityProxy (BaseUpgradeabilityProxy)
  - [Pub] <Constructor> ($)

+  AdminUpgradeabilityProxy (BaseAdminUpgradeabilityProxy,
UpgradeabilityProxy)
  - [Pub] <Constructor> ($)
    - modifiers: UpgradeabilityProxy
  - [Int] _willFallback #

+  __BaseAdminUpgradeabilityProxy__ (BaseUpgradeabilityProxy)
  - [Ext] __admin__
  - [Ext] __implementation__
  - [Ext] __changeAdmin__ #
    - modifiers: ifAdmin
  - [Ext] __upgradeTo__ #
    - modifiers: ifAdmin
  - [Ext] __upgradeToAndCall__ ($)
```

- modifiers: ifAdmin
- [Int] _admin
- [Int] _setAdmin #

+ __AdminUpgradeabilityProxy__ (__BaseAdminUpgradeabilityProxy__, UpgradeabilityProxy)
- [Pub] <Constructor> ($)
- modifiers: UpgradeabilityProxy

+ InitializableUpgradeabilityProxy (BaseUpgradeabilityProxy)
- [Pub] initialize ($)

+ InitializableAdminUpgradeabilityProxy (BaseAdminUpgradeabilityProxy, InitializableUpgradeabilityProxy)
- [Pub] initialize ($)
- [Int] _willFallback #

+ [Int] IProxyFactory
- [Ext] productImplementation
- [Ext] productImplementations

+ ProductProxy (Proxy)
- [Pub] productName
- [Int] _setFactory #
- [Int] _factory
- [Int] _implementation

+ InitializableProductProxy (ProductProxy)
- [Pub] initialize ($)

+ [Lib] OpenZeppelinUpgradesAddress
- [Int] isContract

+ Initializable
  - [Prv] isConstructor

+ ReentrancyGuardUpgradeSafe (Initializable)
  - [Int] __ReentrancyGuard_init #
    - modifiers: initializer
  - [Int] __ReentrancyGuard_init_unchained #
    - modifiers: initializer

+ Governable (Initializable)
  - [Pub] __Governable_init_unchained #
    - modifiers: initializer
  - [Int] _admin
  - [Pub] renounceGovernorship #
    - modifiers: governance
  - [Pub] transferGovernorship #
    - modifiers: governance
  - [Int] _transferGovernorship #

+ Configurable (Governable)
  - [Pub] getConfig
  - [Pub] getConfigI
  - [Pub] getConfigA
  - [Int] _setConfig #
  - [Int] _setConfigI #
  - [Int] _setConfigA #
  - [Ext] setConfig #
    - modifiers: governance
  - [Ext] setConfigI #
    - modifiers: governance
  - [Pub] setConfigA #

```
    - modifiers: governance


+ [Int] IStakingRewards
  - [Ext] lastTimeRewardApplicable
  - [Ext] rewardPerToken
  - [Ext] rewards
  - [Ext] earned
  - [Ext] getRewardForDuration
  - [Ext] totalSupply
  - [Ext] balanceOf
  - [Ext] stake #
  - [Ext] withdraw #
  - [Ext] getReward #
  - [Ext] exit #


+  RewardsDistributionRecipient
  - [Ext] notifyRewardAmount #


+  StakingRewards (IStakingRewards, RewardsDistributionRecipient,
ReentrancyGuardUpgradeSafe)
  - [Pub] __StakingRewards_init #
    - modifiers: initializer
  - [Pub] __StakingRewards_init_unchained #
    - modifiers: initializer
  - [Pub] totalSupply
  - [Pub] balanceOf
  - [Pub] lastTimeRewardApplicable
  - [Pub] rewardPerToken
  - [Pub] earned
  - [Ext] getRewardForDuration
  - [Pub] stakeWithPermit #
    - modifiers: nonReentrant,updateReward
```

```
  - [Pub] stake #
    - modifiers: nonReentrant,updateReward
  - [Pub] withdraw #
    - modifiers: nonReentrant,updateReward
  - [Pub] getReward #
    - modifiers: nonReentrant,updateReward
  - [Pub] exit #
  - [Ext] notifyRewardAmount #
    - modifiers: onlyRewardsDistribution,updateReward

+ [Int] IPermit
  - [Ext] permit #

+  StakingPool (Configurable, StakingRewards)
  - [Pub] __StakingPool_init #
    - modifiers: initializer
  - [Pub] __StakingPool_init_unchained #
    - modifiers: governance
  - [Pub] notifyRewardBegin #
    - modifiers: governance,updateReward
  - [Ext] notifyReward2 #
    - modifiers: governance,updateReward
  - [Pub] rewardDelta
  - [Pub] rewardPerToken
  - [Pub] earned
  - [Pub] getReward #
  - [Pub] getRewardA #
    - modifiers: nonReentrant,updateReward
  - [Ext] getRewardForDuration
  - [Ext] rewards2Token
  - [Ext] rewards2Ratio
```

```
  - [Ext] setPath #
    - modifiers: governance
  - [Pub] lptValueTotal
  - [Pub] lptValue
  - [Pub] swapValue
  - [Pub] TVL
  - [Pub] APY

+ [Int] IUniswapV2Factory
  - [Ext] getPair

+ [Int] IUniswapV2Pair
  - [Ext] getReserves

+ [Int] IWETH (IERC20)
  - [Ext] deposit ($)
  - [Ext] withdraw #

+  EthPool (StakingPool)
  - [Pub] __EthPool_init #
    - modifiers: initializer
  - [Pub] __EthPool_init_unchained #
    - modifiers: governance
  - [Pub] stakeEth ($)
    - modifiers: nonReentrant,updateReward
  - [Pub] withdrawEth #
    - modifiers: nonReentrant,updateReward
  - [Pub] exitEth #
  - [Ext] <Fallback> ($)

+  DoublePool (StakingPool)
```

- [Pub] __DoublePool_init #
  - modifiers: initializer
- [Pub] __DoublePool_init_unchained #
  - modifiers: governance
- [Pub] notifyRewardBegin #
  - modifiers: governance,updateReward2
- [Pub] stake #
  - modifiers: updateReward2
- [Pub] withdraw #
  - modifiers: updateReward2
- [Pub] getReward2 #
  - modifiers: nonReentrant,updateReward2
- [Pub] getDoubleReward #
- [Pub] exit #
- [Pub] rewardPerToken2
- [Pub] earned2

+ [Int] IMasterChef
  - [Ext] poolInfo
  - [Ext] userInfo
  - [Ext] pending
  - [Ext] pendingCake
  - [Ext] deposit #
  - [Ext] withdraw #

+ NestMasterChef (StakingPool)
  - [Pub] __NestMasterChef_init #
    - modifiers: initializer
  - [Pub] __NestMasterChef_init_unchained #
    - modifiers: governance
  - [Pub] notifyRewardBegin #

```
    - modifiers: governance,updateReward2
  - [Pub] migrate #
    - modifiers: governance,updateReward2
  - [Pub] stake #
    - modifiers: updateReward2
  - [Pub] withdraw #
    - modifiers: updateReward2
  - [Pub] getReward2 #
    - modifiers: nonReentrant,updateReward2
  - [Pub] getDoubleReward #
  - [Pub] exit #
  - [Pub] rewardPerToken2
  - [Pub] earned2

+ IioPoolV2 (StakingPool)
  - [Pub] setReward3 #
    - modifiers: governance
  - [Pub] applyReward3 #
    - modifiers: updateReward3
  - [Pub] rewardDelta3
  - [Pub] rewardPerToken3
  - [Pub] earned3
  - [Int] _updateReward3 #
  - [Pub] stake #
  - [Pub] withdraw #
  - [Pub] getReward3 #
    - modifiers: nonReentrant,updateReward3

+ NestMasterChefIioV2 (NestMasterChef, IioPoolV2)
  - [Pub] notifyRewardBegin #
  - [Pub] stake #
```

```
  - [Pub] withdraw #
  - [Pub] exit #

+  BurningPool (StakingPool)
  - [Pub] stake #
  - [Pub] withdraw #

+  Mine (Governable)
  - [Pub] __Mine_init #
    - modifiers: initializer
  - [Pub] __Mine_init_unchained #
    - modifiers: governance
  - [Pub] approvePool #
    - modifiers: governance
  - [Pub] approveToken #
    - modifiers: governance
```

Detailed Results

Issues Checking Status

1. Delegatecall to Untrusted Callee

- SWC ID:112
- Severity: High
- Location: https://bscscan.com/address/0x112Dac520F63fa8Bf6f4B9e922FaccE5E1b320E2#code
- Relationships: CWE-829: Inclusion of Functionality from Untrusted Control Sphere
- Description: The contract delegates execution to another contract with a user-supplied address. The smart contract delegates execution to a user-supplied address. This could allow an attacker to execute arbitrary code in the context of this contract account and manipulate the state of the contract account or execute actions on its behalf.

```
425     function _delegate(address implementation) internal {
426       assembly {
427         // Copy msg.data. We take full control of memory in this inline assembly
428         // block because it will not return to Solidity code. We overwrite the
429         // Solidity scratch pad at memory position 0.
430         calldatacopy(0, 0, calldatasize())
431
432         // Call the implementation.
433         // out and outsize are 0 because we don't know the size yet.
434         let result := delegatecall(gas(), implementation, 0, calldatasize(), 0, 0)
435
436         // Copy the returned data.
437         returndatacopy(0, 0, returndatasize())
438
439         switch result
440         // delegatecall returns 0 on error.
441         case 0 { revert(0, returndatasize()) }
442         default { return(0, returndatasize()) }
443       }
```

- Remediations: Use delegatecall with caution and make sure to never call into untrusted contracts. If the target address is derived from user input ensure to check it against a whitelist of trusted contracts.
- Acknowledgement: After the first phase of Audit, this issue was discussed with the BLACKHOLE PROTOCOL's dev team, and they Acknowledged the issue with this, "the _delete function is an internal function, not an external function, and there is no question of calling uncontrolled contract code."

2. Delegatecall to Untrusted Callee

- SWC ID:112
- Severity: High
- Location: https://bscscan.com/address/0x112Dac520F63fa8Bf6f4B9e922FaccE5E1b320E2#code
- Relationships: CWE-829: Inclusion of Functionality from Untrusted Control Sphere
- Description: The contract delegates execution to another contract with a user-supplied address. The smart contract delegates execution to a user-supplied address.This could allow an attacker to execute arbitrary code in the context of this contract account and manipulate the state of the contract account or execute actions on its behalf..

```
883     function initialize(address _logic, bytes memory _data) public payable {
884       require(_implementation() == address(0));
885       assert(IMPLEMENTATION_SLOT == bytes32(uint256(keccak256('eip1967.proxy.implementation'
886       _setImplementation(_logic);
887       if(_data.length > 0) {
888         (bool success,) = _logic.delegatecall(_data);
889         require(success);
890       }
891     }
```

- Remediations: Use delegatecall with caution and make sure to never call into untrusted contracts. If the target address is derived from user input ensure to check it against a whitelist of trusted contracts.
- Acknowledgement: After the first phase of Audit, this issue was discussed with the BLACKHOLE PROTOCOL's dev team, and they Acknowledged the issue with this, "the initialize function is an initialization function, which will only be called once by the deployer during deployment, and there is no problem of calling uncontrolled contract code."

3. Reentrancy

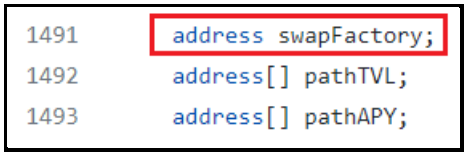
- SWC ID:107
- Severity: Medium
- Location: https://bscscan.com/address/0x112Dac520F63fa8Bf6f4B9e922FaccE5E1b320E2#code
- Relationships: CWE-841: Improper Enforcement of Behavioral Workflow
- Description: Write to persistent state following external call. The contract account state is accessed after an external call to a user defined address. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

```
624     function _setAdmin(address newAdmin) internal {
625       bytes32 slot = ADMIN_SLOT;
626
627       assembly {
628         sstore(slot, newAdmin)
629       }
630     }
```

- Remediations: The best practices to avoid Reentrancy weaknesses are:
  - Make sure all internal state changes are performed before the call is executed. This is known as the Checks-Effects-Interactions pattern
  - Use a reentrancy lock (ie. OpenZeppelin's ReentrancyGuard.
- Acknowledgement: After the first phase of Audit, this issue was discussed with the BLACKHOLE PROTOCOL's dev team, and they Acknowledged the issue with, "_setAdmin is an internal function and there is no re-entry problem."

4. State Variable Default Visibility

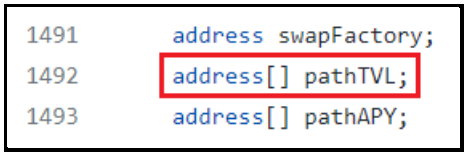
- SWC ID:108
- Severity: Low
- Location: https://bscscan.com/address/0x112Dac520F63fa8Bf6f4B9e922FaccE5E1b320E2#code
- Relationships: CWE-710: Improper Adherence to Coding Standards
- Description: State variable visibility is not set. It is best practice to set the visibility of state variables explicitly. The default visibility for "swapFactory" is internal. Other possible visibility settings are public and private.



- Remediations: Variables can be specified as being public, internal or private. Explicitly define visibility for all state variables.
- Acknowledgement: After the first phase of Audit, this issue was discussed with the BLACKHOLE PROTOCOL's dev team, and they Acknowledged the issue with, "these three variables are internally accessible by default, there is no security issue."

5. **State Variable Default Visibility**

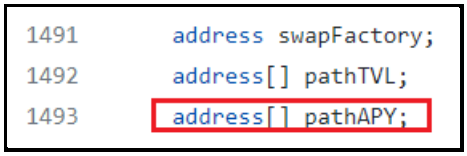
- SWC ID:108
- Severity: Low
- Location: https://bscscan.com/address/0x112Dac520F63fa8Bf6f4B9e922FaccE5E1b320E2#code
- Relationships: CWE-710: Improper Adherence to Coding Standards
- Description: State variable visibility is not set. It is best practice to set the visibility of state variables explicitly. The default visibility for "pathTVL" is internal. Other possible visibility settings are public and private.



- Remediations: Variables can be specified as being public, internal or private. Explicitly define visibility for all state variables.
- Acknowledgement: After the first phase of Audit, this issue was discussed with the BLACKHOLE PROTOCOL's dev team, and they Acknowledged the issue with, "these three variables are internally accessible by default, there is no security issue."

6. State Variable Default Visibility

- SWC ID:108
- Severity: Low
- Location: https://bscscan.com/address/0x112Dac520F63fa8Bf6f4B9e922FaccE5E1b320E2#code
- Relationships: CWE-710: Improper Adherence to Coding Standards
- Description: State variable visibility is not set. It is best practice to set the visibility of state variables explicitly. The default visibility for "pathAPY" is internal. Other possible visibility settings are public and private.



- Remediations: Variables can be specified as being public, internal or private. Explicitly define visibility for all state variables.
- Acknowledgement: After the first phase of Audit, this issue was discussed with the BLACKHOLE PROTOCOL's dev team, and they Acknowledged the issue with, "these three variables are internally accessible by default, there is no security issue."

7. Reentrancy

- SWC ID:107
- Severity: Low
- Location: https://bscscan.com/address/0x112Dac520F63fa8Bf6f4B9e922FaccE5E1b320E2#code
- Relationships: CWE-841: Improper Enforcement of Behavioral Workflow
- Description: A call to a user-supplied address is executed. An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Reentering the contract in an intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or reentrancy guards are in place.

```
2190
2191        function approvePool(address pool, uint amount) public governance {
2192            IERC20(reward).approve(pool, amount);
2193        }
2194
2195        function approveToken(address token, address pool, uint amount) public governa
2196            IERC20(token).approve(pool, amount);
2197        }
2198
```

- Remediations: The best practices to avoid Reentrancy weaknesses are:
  - Make sure all internal state changes are performed before the call is executed. This is known as the Checks-Effects-Interactions pattern
  - Use a reentrancy lock (ie. OpenZeppelin's ReentrancyGuard.
- Acknowledgement: After the first phase of Audit, this issue was discussed with the BLACKHOLE PROTOCOL's dev team, and they Acknowledged the issue with, "approvePool can only be called by the administrator, there is no re-entry problem."

8. Reentrancy

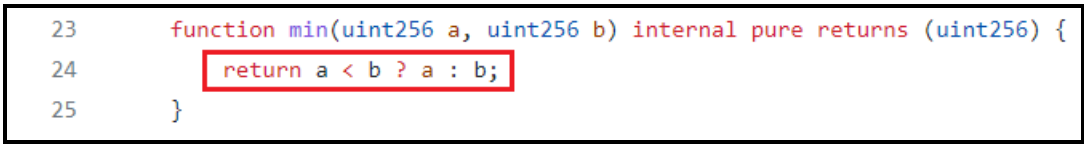
- SWC ID:107
- Severity: Low
- Location: https://bscscan.com/address/0x112Dac520F63fa8Bf6f4B9e922FaccE5E1b320E2#code
- Relationships: CWE-841: Improper Enforcement of Behavioral Workflow
- Description: A call to a user-supplied address is executed. An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Reentering the contract in an intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or reentrancy guards are in place.

```
2190
2191        function approvePool(address pool, uint amount) public governance {
2192            IERC20(reward).approve(pool, amount);
2193        }
2194
2195        function approveToken(address token, address pool, uint amount) public gove
2196            IERC20(token).approve(pool, amount);
2197        }
2198
```

- Remediations: The best practices to avoid Reentrancy weaknesses are:
  - Make sure all internal state changes are performed before the call is executed. This is known as the Checks-Effects-Interactions pattern
  - Use a reentrancy lock (ie. OpenZeppelin's ReentrancyGuard.
- Acknowledgement: After the first phase of Audit, this issue was discussed with the BLACKHOLE PROTOCOL's dev team, and they Acknowledged the issue with, "approveToken can only be called by the administrator, there is no re-entry problem."

9. **Block values as a proxy for time**

- SWC ID:116
- Severity: Low
- Location: https://bscscan.com/address/0x112Dac520F63fa8Bf6f4B9e922FaccE5E1b320E2#code
- Relationships: CWE-829: Inclusion of Functionality from Untrusted Control Sphere
- Description: A control flow decision is made based on The block.timestamp environment variable. The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.



- Remediations: Developers should write smart contracts with the notion that block values are not precise, and the use of them can lead to unexpected effects. Alternatively, they may make use oracles.
- Acknowledgement: After the first phase of Audit, this issue was discussed with the BLACKHOLE PROTOCOL's dev team, and they Acknowledged the issue with, "this line of code does not involve any security problems"

10. Block values as a proxy for time

- SWC ID:116
- Severity: Low
- Location: https://bscscan.com/address/0x112Dac520F63fa8Bf6f4B9e922FaccE5E1b320E2#code
- Relationships: CWE-829: Inclusion of Functionality from Untrusted Control Sphere
- Description: A control flow decision is made based on The block.timestamp environment variable. The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

```
1426      function notifyRewardAmount(uint256 reward) override external onlyRewardsDistr
1427          if (block.timestamp >= periodFinish) {
1428              rewardRate = reward.div(rewardsDuration);
1429          } else {
1430              uint256 remaining = periodFinish.sub(block.timestamp);
1431              uint256 leftover = remaining.mul(rewardRate);
1432              rewardRate = reward.add(leftover).div(rewardsDuration);
1433          }
```

- Remediations: Developers should write smart contracts with the notion that block values are not precise, and the use of them can lead to unexpected effects. Alternatively, they may make use oracles.
- Acknowledgement: After the first phase of Audit, this issue was discussed with the BLACKHOLE PROTOCOL's dev team, and they Acknowledged the issue with, "the production of mining is calculated by time, a small time deviation does not affect the business logic."

11. Block values as a proxy for time

- SWC ID:116
- Severity: Low
- Location: https://bscscan.com/address/0x112Dac520F63fa8Bf6f4B9e922FaccE5E1b320E2#code
- Relationships: CWE-829: Inclusion of Functionality from Untrusted Control Sphere
- Description: A control flow decision is made based on The block.timestamp environment variable. The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

```
1684      function APY() virtual public view returns (uint) {
1685          uint amt = rewardsToken.allowance(rewardsDistribution, addr
1686
1687          if(lep == 3) {
1688              uint amt2 = amt.mul(365 days).mul(now.add(rewardsDurati
1689              amt = amt.sub(amt2);
1690          } else if(lep == 2) {
1691              amt = amt.mul(365 days).div(rewardsDuration);
1692          }else if(now < periodFinish)
1693              amt = amt.mul(365 days).div(periodFinish.sub(lastUpdate
1694          else if(lastUpdateTime >= periodFinish)
1695              amt = 0;
1696
1697          require(address(rewardsToken) == pathAPY[0]);
1698          amt = swapValue(amt, pathAPY);
1699          return amt.mul(1e18).div(TVL());
1700      }
```

- Remediations: Developers should write smart contracts with the notion that block values are not precise, and the use of them can lead to unexpected effects. Alternatively, they may make use oracles.
- Acknowledgement: After the first phase of Audit, this issue was discussed with the BLACKHOLE PROTOCOL's dev team, and they Acknowledged the issue with, "the production of mining is calculated by time, a small time deviation does not affect the business logic."

12. Requirement Violation

- SWC ID:123
- Severity: Low
- Location: https://bscscan.com/address/0x112Dac520F63fa8Bf6f4B9e922FaccE5E1b320E2#code
- Relationships: CWE-573: Improper Following of Specification by Caller.
- Description: Requirement violation. A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

```
425     function _delegate(address implementation) internal {
426       assembly {
427         // Copy msg.data. We take full control of memory in this inline assembly
428         // block because it will not return to Solidity code. We overwrite the
429         // Solidity scratch pad at memory position 0.
430         calldatacopy(0, 0, calldatasize())
431
432         // Call the implementation.
433         // out and outsize are 0 because we don't know the size yet.
434         let result := delegatecall(gas(), implementation, 0, calldatasize(), 0, 0)
435
436         // Copy the returned data.
437         returndatacopy(0, 0, returndatasize())
438
439         switch result
440         // delegatecall returns 0 on error.
441         case 0 { revert(0, returndatasize()) }
442         default { return(0, returndatasize()) }
443       }
444     }
```

- Remediations: If the required logical condition is too strong, it should be weakened to allow all valid external inputs. Otherwise, the bug must be in the contract that provided the external input and one should consider fixing its code by making sure no invalid inputs are provided.
- Acknowledgement: After the first phase of Audit, this issue was discussed with the BLACKHOLE PROTOCOL's dev team, and they Acknowledged the issue with, "there is no nested calls, no problem."

13. Requirement Violation

- SWC ID:123
- Severity: Low
- Location: https://bscscan.com/address/0x112Dac520F63fa8Bf6f4B9e922FaccE5E1b320E2#code
- Relationships: CWE-573: Improper Following of Specification by Caller.
- Description: Requirement violation. A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

```
1471  contract StakingPool is Configurable, StakingRewards {
1472      using Address for address payable;
1473
1474      bytes32 internal constant _ecoAddr_         = 'ecoAddr';
1475      bytes32 internal constant _ecoRatio_        = 'ecoRatio';
1476          bytes32 internal constant _allowContract_   = 'allowContract';
1477          bytes32 internal constant _allowlist_       = 'allowlist';
1478          bytes32 internal constant _blocklist_       = 'blocklist';
1479
1480          bytes32 internal constant _rewards2Token_   = 'rewards2Token';
1481          bytes32 internal constant _rewards2Ratio_   = 'rewards2Ratio';
1482          //bytes32 internal constant _rewards2Span_    = 'rewards2Span';
1483          bytes32 internal constant _rewards2Begin_   = 'rewards2Begin';
```

- Remediations: If the required logical condition is too strong, it should be weakened to allow all valid external inputs. Otherwise, the bug must be in the contract that provided the external input and one should consider fixing its code by making sure no invalid inputs are provided.
- Acknowledgement: After the first phase of Audit, this issue was discussed with the BLACKHOLE PROTOCOL's dev team, and they Acknowledged the issue with, "there is no nested calls, no problem."

14. Requirement Violation

- SWC ID:123
- Severity: Low
- Location: https://bscscan.com/address/0x112Dac520F63fa8Bf6f4B9e922FaccE5E1b320E2#code
- Relationships: CWE-573: Improper Following of Specification by Caller.
- Description: Requirement violation. A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

```
1565    function earned(address account) virtual override public view returns (uint256) {
1566        return Math.min(Math.min(super.earned(account), rewardsToken.allowance(rewardsDistribution, address(this))), rewardsToken.balanceOf(rewardsDistribution));
1567        }
1568
```

- Remediations: If the required logical condition is too strong, it should be weakened to allow all valid external inputs. Otherwise, the bug must be in the contract that provided the external input and one should consider fixing its code by making sure no invalid inputs are provided.
- Acknowledgement: After the first phase of Audit, this issue was discussed with the BLACKHOLE PROTOCOL's dev team, and they Acknowledged the issue with, "there is no nested calls, no problem."

15. Requirement Violation

- SWC ID:123
- Severity: Low
- Location: https://bscscan.com/address/0x112Dac520F63fa8Bf6f4B9e922FaccE5E1b320E2#code
- Relationships: CWE-573: Improper Following of Specification by Caller.
- Description: Requirement violation. A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

```
1719  contract EthPool is StakingPool {
1720      bytes32 internal constant _WETH_                    = 'WETH';
1721
1722      function __EthPool_init(address _governor,
1723          address _rewardsDistribution,
1724          address _rewardsToken,
1725          address _stakingToken,
1726          address _ecoAddr,
1727                  address _WETH
1728      ) public virtual initializer {
1729              __ReentrancyGuard_init_unchained();
1730              __Governable_init_unchained(_governor);
1731          //__StakingRewards_init_unchained(_rewardsDistribution, _rewardsToken, _stakingT
1732          __StakingPool_init_unchained(_rewardsDistribution, _rewardsToken, _stakingToken,
1733                  __EthPool_init_unchained(_WETH);
1734      }
```

- Remediations: If the required logical condition is too strong, it should be weakened to allow all valid external inputs. Otherwise, the bug must be in the contract that provided the external input and one should consider fixing its code by making sure no invalid inputs are provided.
- Acknowledgement: After the first phase of Audit, this issue was discussed with the BLACKHOLE PROTOCOL's dev team, and they Acknowledged the issue with, "there is no nested calls, no problem."

16. Requirement Violation

- SWC ID:123
- Severity: Low
- Location: https://bscscan.com/address/0x112Dac520F63fa8Bf6f4B9e922FaccE5E1b320E2#code
- Relationships: CWE-573: Improper Following of Specification by Caller.
- Description: Requirement violation. A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

```
1956  contract IioPoolV2 is StakingPool {          // support multi IIO at the same time
1957      //address internal constant HelmetAddress = 0x948d2a81086A075b3130BAc19e4c6DE
1958      address internal constant BurnAddress    = 0x0000000000000000000000000000000000000000
```

- Remediations: If the required logical condition is too strong, it should be weakened to allow all valid external inputs. Otherwise, the bug must be in the contract that provided the external input and one should consider fixing its code by making sure no invalid inputs are provided.
- Acknowledgement: After the first phase of Audit, this issue was discussed with the BLACKHOLE PROTOCOL's dev team, and they Acknowledged the issue with, "there is no nested calls, no problem."

17. Requirement Violation

- SWC ID:123
- Severity: Low
- Location: https://bscscan.com/address/0x112Dac520F63fa8Bf6f4B9e922FaccE5E1b320E2#code
- Relationships: CWE-573: Improper Following of Specification by Caller.
- Description: Requirement violation. A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

```
2160  contract BurningPool is StakingPool {
2161      address internal constant BurnAddress    = 0x000000000000000000000000000000
2162
2163      function stake(uint256 amount) virtual override public {
2164          super.stake(amount);
2165          stakingToken.safeTransfer(BurnAddress, stakingToken.balanceOf(address
2166      }
2167
2168      function withdraw(uint256) virtual override public {
2169          revert('Burned already, none to withdraw');
2170      }
2171
2172      // Reserved storage space to allow for layout changes in the future.
2173      uint256[50] private ______gap;
2174  }
2175
```

- Remediations: If the required logical condition is too strong, it should be weakened to allow all valid external inputs. Otherwise, the bug must be in the contract that provided the external input and one should consider fixing its code by making sure no invalid inputs are provided.
- Acknowledgement: After the first phase of Audit, this issue was discussed with the BLACKHOLE PROTOCOL's dev team, and they Acknowledged the issue with, "there is no nested calls, no problem."

18. Requirement Violation

- SWC ID:123
- Severity: Low
- Location: https://bscscan.com/address/0x112Dac520F63fa8Bf6f4B9e922FaccE5E1b320E2#code
- Relationships: CWE-573: Improper Following of Specification by Caller.
- Description: Requirement violation. A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

```
2177  contract Mine is Governable {
2178      using SafeERC20 for IERC20;
2179
2180      address public reward;
2181
2182      function __Mine_init(address governor, address reward_) public initializer
2183          __Governable_init_unchained(governor);
2184          __Mine_init_unchained(reward_);
2185      }
2186
2187      function __Mine_init_unchained(address reward_) public governance {
2188          reward = reward_;
2189      }
2190
2191      function approvePool(address pool, uint amount) public governance {
2192          IERC20(reward).approve(pool, amount);
2193      }
2194
2195      function approveToken(address token, address pool, uint amount) public gove
2196          IERC20(token).approve(pool, amount);
2197      }
2198
```

- Remediations: If the required logical condition is too strong, it should be weakened to allow all valid external inputs. Otherwise, the bug must be in the contract that provided the external input and one should consider fixing its code by making sure no invalid inputs are provided.
- Acknowledgement: After the first phase of Audit, this issue was discussed with the BLACKHOLE PROTOCOL's dev team, and they Acknowledged the issue with, "there is no nested calls, no problem."

19. DoS with Failed Call

- SWC ID:113
- Severity: Low
- Location: https://bscscan.com/address/0x112Dac520F63fa8Bf6f4B9e922FaccE5E1b320E2#code
- Relationships: CWE-703: Improper Check or Handling of Exceptional Conditions
- Description: Multiple calls are executed in the same transaction. This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

```
425     function _delegate(address implementation) internal {
426       assembly {
427         // Copy msg.data. We take full control of memory in this inline assembly
428         // block because it will not return to Solidity code. We overwrite the
429         // Solidity scratch pad at memory position 0.
430         calldatacopy(0, 0, calldatasize())
431
432         // Call the implementation.
433         // out and outsize are 0 because we don't know the size yet.
434         let result := delegatecall(gas(), implementation, 0, calldatasize(), 0, 0)
435
436         // Copy the returned data.
437         returndatacopy(0, 0, returndatasize())
438
439         switch result
440         // delegatecall returns 0 on error.
441         case 0 { revert(0, returndatasize()) }
442         default { return(0, returndatasize()) }
443       }
444     }
```

- Remediations: It is recommended to follow call best practices:
  - Avoid combining multiple calls in a single transaction, especially when calls are executed as part of a loop
  - Always assume that external calls can fail
  - Implement the contract logic to handle failed calls
- Acknowledgement: After the first phase of Audit, this issue was discussed with the BLACKHOLE PROTOCOL's dev team, and they Acknowledged the issue with, "codes are OpenZeppelin's proxy library, there is no exception checking or mishandling."

20. DoS with Failed Call

- SWC ID:113
- Severity: Low
- Location: https://bscscan.com/address/0x112Dac520F63fa8Bf6f4B9e922FaccE5E1b320E2#code
- Relationships: CWE-703: Improper Check or Handling of Exceptional Conditions
- Description: Multiple calls are executed in the same transaction. This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

```
1003    function initialize(address factory, bytes memory data) public payable {
1004      require(_factory() == address(0));
1005      assert(FACTORY_SLOT == bytes32(uint256(keccak256('eip1967.proxy.factory
1006      _setFactory(factory);
1007      if(data.length > 0) {
1008        (bool success,) = _implementation().delegatecall(data);
1009        require(success);
1010      }
1011    }
1012  }
```

- Remediations: It is recommended to follow call best practices:
  - Avoid combining multiple calls in a single transaction, especially when calls are executed as part of a loop
  - Always assume that external calls can fail
  - Implement the contract logic to handle failed calls
- Acknowledgement: After the first phase of Audit, this issue was discussed with the BLACKHOLE PROTOCOL's dev team, and they Acknowledged the issue with, "codes are OpenZeppelin's proxy library, there is no exception checking or mishandling."

Basic Coding Bugs

1. **Constructor Mismatch**
   - Description: Whether the contract name and its constructor are not identical to each other.
   - Result: PASSED
   - Severity: Critical
2. **Ownership Takeover**
   - Description: Whether the set owner function is not protected.
   - Result: PASSED
   - Severity: Critical
3. **Redundant Fallback Function**
   - Description: Whether the contract has a redundant fallback function.
   - Result: PASSED
   - Severity: Critical
4. **Overflows & Underflows**
   - Description: Whether the contract has general overflow or underflow vulnerabilities
   - Result: PASSED
   - Severity: Critical
5. **Reentrancy**
   - Description: Reentrancy is an issue when code can call back into your contract and change state, such as withdrawing ETHs.
   - Result: PASSED
   - Severity: Critical
6. **MONEY-Giving Bug**
   - Description: Whether the contract returns funds to an arbitrary address.
   - Result: PASSED
   - Severity: High

7. **Blackhole**

   - Description: Whether the contract locks ETH indefinitely: merely in without out.
   - Result: PASSED
   - Severity: High

8. **Unauthorized Self-Destruct**

   - Description: Whether the contract can be killed by any arbitrary address.
   - Result: PASSED
   - Severity: Medium

9. **Revert DoS**

   - Description: Whether the contract is vulnerable to DoS attack because of unexpected revert.
   - Result: PASSED
   - Severity: Medium

10. **Unchecked External Call**

    - Description: Whether the contract has any external call without checking the return value.
    - Result: PASSED
    - Severity: Medium

11. **Gasless Send**

    - Description: Whether the contract is vulnerable to gasless send.
    - Result: PASSED
    - Severity: Medium

12. **Send Instead of Transfer**

    - Description: Whether the contract uses send instead of transfer.
    - Result: PASSED
    - Severity: Medium

13. **Costly Loop**

- Description: Whether the contract has any costly loop which may lead to Out-Of-Gas exception.
- Result: PASSED
- Severity: Medium

14. **(Unsafe) Use of Untrusted Libraries**

- Description: Whether the contract use any suspicious libraries.
- Result: PASSED
- Severity: Medium

15. **(Unsafe) Use of Predictable Variables**

- Description: Whether the contract contains any randomness variable, but its value can be predicated.
- Result: PASSED
- Severity: Medium

16. **Transaction Ordering Dependence**

- Description: Whether the final state of the contract depends on the order of the transactions.
- Result: PASSED
- Severity: Medium

17. **Deprecated Uses**

- Description: Whether the contract use the deprecated tx.origin to perform the authorization.
- Result: PASSED
- Severity: Medium

Semantic Consistency Checks

- Description: Whether the semantic of the white paper is different from the implementation of the contract.
- Result: PASSED
- Severity: Critical

Conclusion

In this audit, we thoroughly analyzed BLACKHOLE PROTOCOL's Smart Contract. The current code base is well organized but there are promptly some High, Medium and low issues found in the first phase of Smart Contract Audit, which is acknowledged by BLACKHOLE PROTOCOL's dev team but as the code was written with understanding that any of the issues could not be harmful for the contract for any safety related issues and also, there is no such serious or performance issues so, they've decided to remain the code unchanged.

Meanwhile, we need to emphasize that smart contracts as a whole are still in an early, but exciting stage of development. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

About eNebula Solutions

We believe that people have a fundamental need to security and that the use of secure solutions enables every person to more freely use the Internet and every other connected technology. We aim to provide security consulting service to help others make their solutions more resistant to unauthorized access to data & inadvertent manipulation of the system. We support teams from the design phase through the production to launch and surely after.

The eNebula Solutions team has skills for reviewing code in C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, and JavaScript for common security vulnerabilities & specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture, including in cryptocurrency, blockchains, payments, and smart contracts. Additionally, the team can utilize various tools to scan code & networks and build custom tools as necessary.

Although we are a small team, we surely believe that we can have a momentous impact on the world by being translucent and open about the work we do.

For more information about our security consulting,
please mail us at – contact@enebula.in