



Security Assessment

Purple Wave Finance

Verified on 06/24/2023

SUMMARY

Project

Purple Wave Finance

CHAIN

Polygon

METHODOLOGY

Manual & Automatic Analysis

FILES

Single

DELIVERY

06/24/2023

TYPE

Standard Audit



3

0

0

1

0

2

Total Findings

Critical

Major

Medium

Minor

Informational

 0 Critical

0 Pending

An exposure that can affect the contract functions in several events that can risk and disrupt the contract

 0 Major

0 Pending

An exposure that can affect the outcome when using the contract that can serve as an opening in manipulating the contract in an unwanted manner

 0 Medium

1 Pending

An opening that could affect the outcome in executing the contract in a specific situation

 0 Minor

0 Pending

An opening but doesn't have an impact on the functionality of the contract

 5 Informational

2 Pending

An opening that consists information but will not risk or affect the contract

STATUS
 **AUDIT PASSED**

TABLE OF CONTENTS | Purple Wave Finance

| Summary

Project Summary
Findings Summary
Disclaimer
Scope of Work
Auditing Approach

| Project Information

Token/Project Details
Inheritance Graph
Call Graph

| Findings

Issues
SWC Attacks
CW Assessment
Fixes & Recommendation
Audit Comments

DISCLAIMER | Purple Wave Finance

ContractWolf audits and reports should not be considered as a form of project's "Advertisement" and does not cover any interaction and assessment from "Project Contract" to "External Contracts" such as PancakeSwap, UniSwap, SushiSwap or similar.

ContractWolf does not provide any warranty on its released report and should not be used as a decision to invest into audited projects.

ContractWolf provides a transparent report to all its "Clients" and to its "Clients Participants" and will not claim any guarantee of bug-free code within its **SMART CONTRACT**.

ContractWolf's presence is to analyze, audit and assess the Client's Smart Contract to find any underlying risk and to eliminate any logic and flow errors within its code.

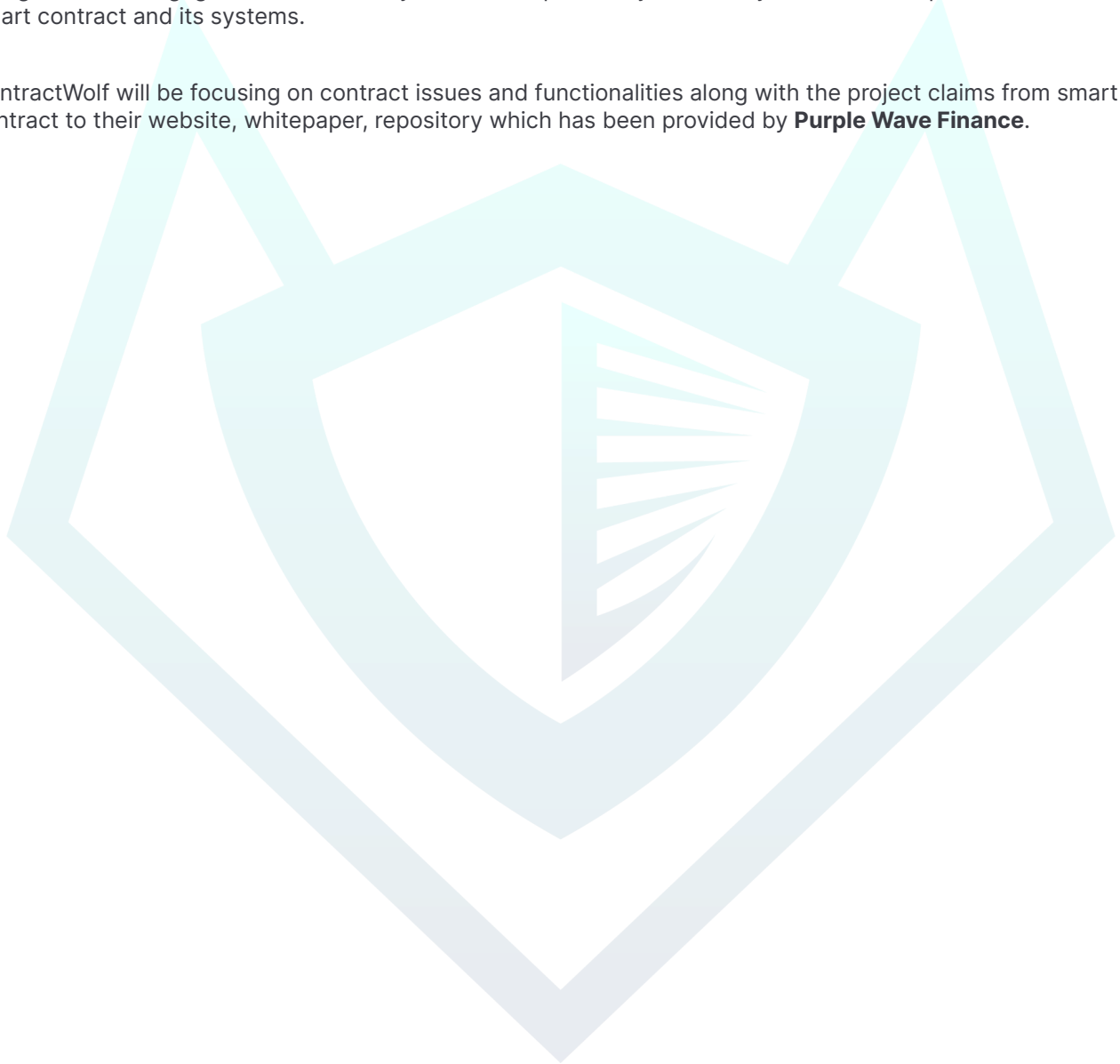
Each company or project should be liable to its security flaws and functionalities.

SCOPE OF WORK | Purple Wave Finance

Purple Wave Finance team has agreed and provided us with the files that need to be tested (*Github, BSCscan, Etherscan, Local files etc*). The scope of audit is the main contract.

The goal of this engagement is to identify if there is a possibility of security flaws in the implementation of smart contract and its systems.

ContractWolf will be focusing on contract issues and functionalities along with the project claims from smart contract to their website, whitepaper, repository which has been provided by **Purple Wave Finance**.



AUDITING APPROACH | Purple Wave Finance

Every line of code along with its functionalities will undergo manual review to check for security issues, quality of logic and contract scope of inheritance. The manual review will be done by our team that will document any issues that they discovered.

METHODOLOGY

The auditing process follows a routine series of steps :

1. Code review that includes the following :
 - Review of the specifications, sources and instructions provided to ContractWolf to make sure we understand the size, scope and functionality of the smart contract.
 - Manual review of code. Our team will have a process of reading the code line-by-line with the intention of identifying potential vulnerabilities, underlying and hidden security flaws.
2. Testing and automated analysis that includes :
 - Testing the smart contract function with common test cases and scenarios to ensure that it returns the expected results.
3. Best practices and ethical review. The team will review the contract with the aim to improve efficiency, effectiveness, clarifications, maintainability, security and control within the smart contract.
4. Recommendations to help the project take steps to eliminate or minimize threats and secure the smart contract.

TOKEN DETAILS | Purple Wave Finance



A safe place to optimize your earnings. Revenue generated from fees goes back to Brainy Budz NFT holders for life. Created by Purple Wave Studios

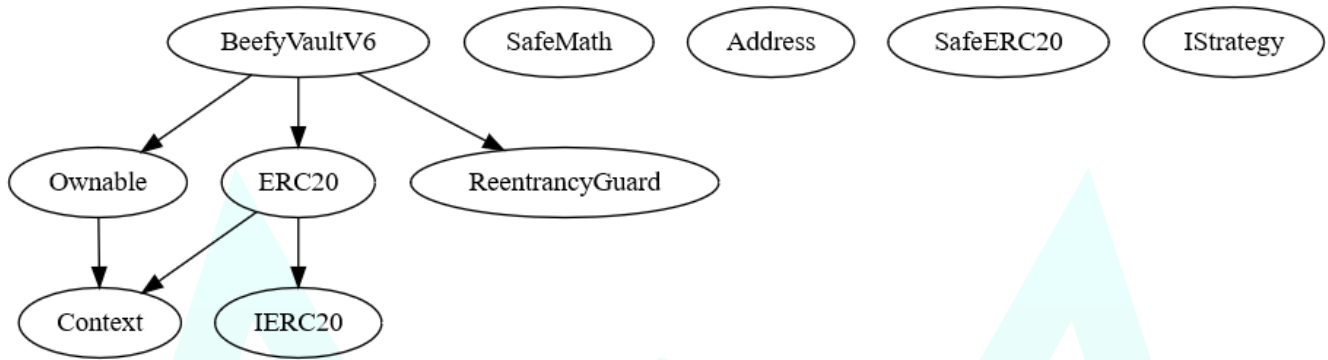
| Token Name | Symbol | Decimal | Total Supply | Chain |
|------------|--------|---------|--------------|---------|
| – | – | 18 | – | Polygon |

SOURCE

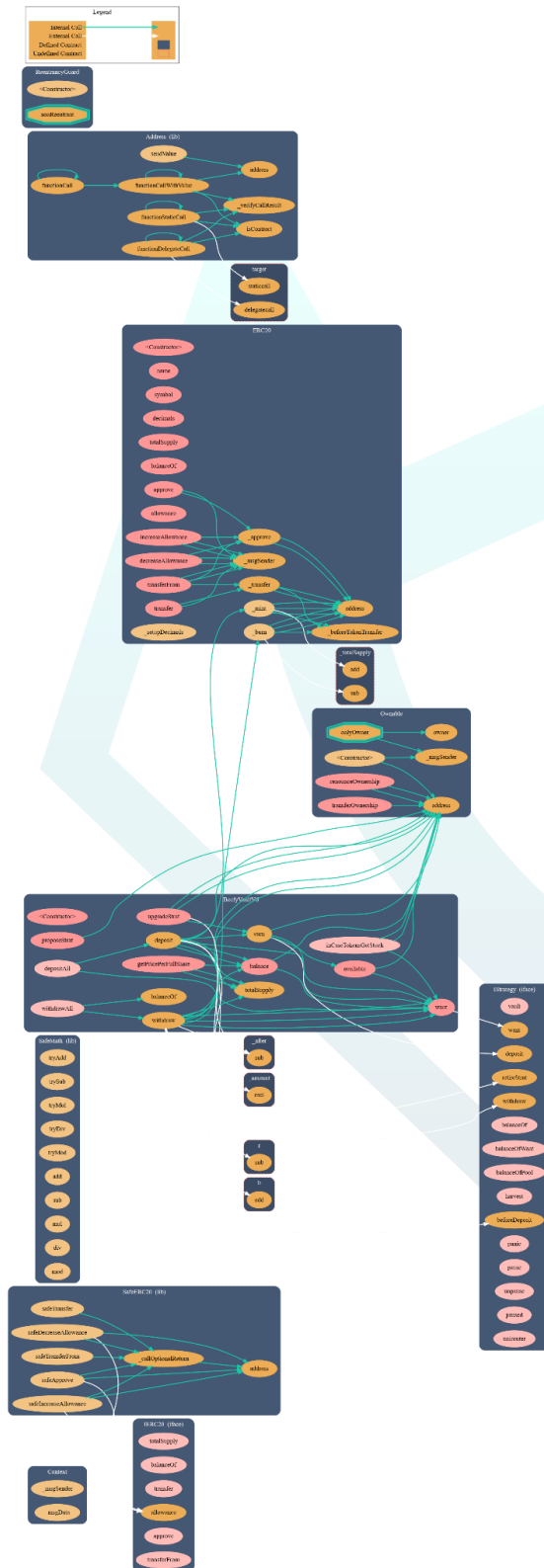
Source <https://polygonscan.com/address/0x059f710a4c28bD66B624C99Ff06b0Ac5205DA791>

INHERITANCE GRAPH | Purple Wave Finance

Inheritance Graph of Contract Functions



Call Graph of Contract Functions



FINDINGS | Purple Wave Finance



3

0

0

1

0

2

Total Findings

Critical

Major

Medium

Minor

Informational

This report has been prepared to state the issues and vulnerabilities for Purple Wave Finance through this audit. The goal of this report findings is to identify specifically and fix any underlying issues and errors

| ID | Title | File & Line # | Severity | Status |
|---------|--------------------------|---|---------------|-----------|
| SWC-103 | Floating Pragma is set | BeefyVaultV6.sol, L : 9, 35, 114, 330, 828, 904, 973 | Informational | • Pending |
| SWC-135 | Code with no Effects | BeefyVaultV6.sol, L : 687, 713, 738, 788, 764 | Informational | • Pending |
| CW-002 | Incorrect Access Control | BeefyVaultV6.sol, L : 1236, 1243 | Medium | • Pending |

SWC ATTACKS | Purple Wave Finance

Smart Contract Weakness Classification and Test Cases

| ID | Description | Status |
|---------|---|--------------|
| SWC-100 | Function Default Visibility | ● Passed |
| SWC-101 | Integer Overflow and Underflow | ● Passed |
| SWC-102 | Outdated Compiler Version | ● Not Passed |
| SWC-103 | Floating Pragma | ● Not Passed |
| SWC-104 | Unchecked Call Return Value | ● Passed |
| SWC-105 | Unprotected Ether Withdrawal | ● Passed |
| SWC-106 | Unprotected SELF DESTRUCT Instruction | ● Passed |
| SWC-107 | Reentrancy | ● Passed |
| SWC-108 | State Variable Default Visibility | ● Passed |
| SWC-109 | Uninitialized Storage Pointer | ● Passed |
| SWC-110 | Assert Violation | ● Passed |
| SWC-111 | Use of Deprecated Solidity Functions | ● Passed |
| SWC-112 | Delegatecall to Untrusted Callee | ● Passed |
| SWC-113 | DoS with Failed Call | ● Passed |
| SWC-114 | Transaction Order Dependence | ● Passed |
| SWC-115 | Authorization through tx.origin | ● Passed |
| SWC-116 | Block values as a proxy for time | ● Passed |
| SWC-117 | Signature Malleability | ● Passed |
| SWC-118 | Incorrect Constructor Name | ● Passed |
| SWC-119 | Shadowing State Variables | ● Passed |
| SWC-120 | Weak Sources of Randomness from Chain Attributes | ● Passed |
| SWC-121 | Missing Protection against Signature Replay Attacks | ● Passed |
| SWC-122 | Lack of Proper Signature Verification | ● Passed |

| ID | Description | Status |
|---------|--|--------------|
| SWC-123 | Requirement Violation | ● Passed |
| SWC-124 | Write to Arbitrary Storage Location | ● Passed |
| SWC-125 | Incorrect Inheritance Order | ● Passed |
| SWC-126 | Insufficient Gas Griefing | ● Passed |
| SWC-127 | Arbitrary Jump with Function Type Variable | ● Passed |
| SWC-128 | DoS With Block Gas Limit | ● Passed |
| SWC-129 | Typographical Error | ● Passed |
| SWC-130 | Right-To-Left-Override control character(U+202E) | ● Passed |
| SWC-131 | Presence of unused variables | ● Passed |
| SWC-132 | Unexpected Ether balance | ● Passed |
| SWC-133 | Hash Collisions With Multiple Variable Arguments | ● Passed |
| SWC-134 | Message call with hardcoded gas amount | ● Passed |
| SWC-135 | Code With No Effects | ● Not Passed |
| SWC-136 | Unencrypted Private Data On-Chain | ● Passed |

CW ASSESSMENT | Purple Wave Finance

ContractWolf Vulnerability and Security Tests

| ID | Name | Description | Status |
|--------|--------------------------|--|--------|
| CW-001 | Multiple Version | Presence of multiple compiler version across all contracts | ✗ |
| CW-002 | Incorrect Access Control | Additional checks for critical logic and flow | ✗ |
| CW-003 | Payable Contract | A function to withdraw ether should exist otherwise the ether will be trapped | ✓ |
| CW-004 | Custom Modifier | major recheck for custom modifier logic | ✓ |
| CW-005 | Divide Before Multiply | Performing multiplication before division is generally better to avoid loss of precision | ✓ |
| CW-006 | Multiple Calls | Functions with multiple internal calls | ✓ |
| CW-007 | Deprecated Keywords | Use of deprecated functions/operators such as block.blockhash() for blockhash(), msg.gas for gasleft(), throw for revert(), sha3() for keccak256(), callcode() for delegatecall(), suicide() for selfdestruct(), constant for view or var for actual type name should be avoided to prevent unintended errors with newer compiler versions | ✓ |
| CW-008 | Unused Contract | Presence of an unused, unimported or uncalled contract | ✓ |
| CW-009 | Assembly Usage | Use of EVM assembly is error-prone and should be avoided or double-checked for correctness | ✓ |
| CW-010 | Similar Variable Names | Variables with similar names could be confused for each other and therefore should be avoided | ✓ |
| CW-011 | Commented Code | Removal of commented/unused code lines | ✓ |
| CW-012 | SafeMath Override | SafeMath is no longer needed starting Solidity v0.8+. The compiler now has Built in overflow checking. | ✓ |

FIXES & RECOMMENDATION

CW-002 | Incorrect Access Control

The logic from the function below might return an error

```
function upgradeStrat() public onlyOwner {  
    require(stratCandidate.implementation != address(0), "There is no candidate");  
    require(stratCandidate.proposedTime.add(approvalDelay) < block.timestamp,  
"Delay has not passed");  
  
    emit UpgradeStrat(stratCandidate.implementation);  
  
    strategy.retireStrat();  
    strategy = IStrategy(stratCandidate.implementation);  
    stratCandidate.implementation = address(0);  
    stratCandidate.proposedTime = 5000000000;  
  
    earn();  
}
```

Line

```
require(stratCandidate.implementation != address(0), "There is no candidate");
```

And line

```
    stratCandidate.implementation = address(0);
```

Recommendation

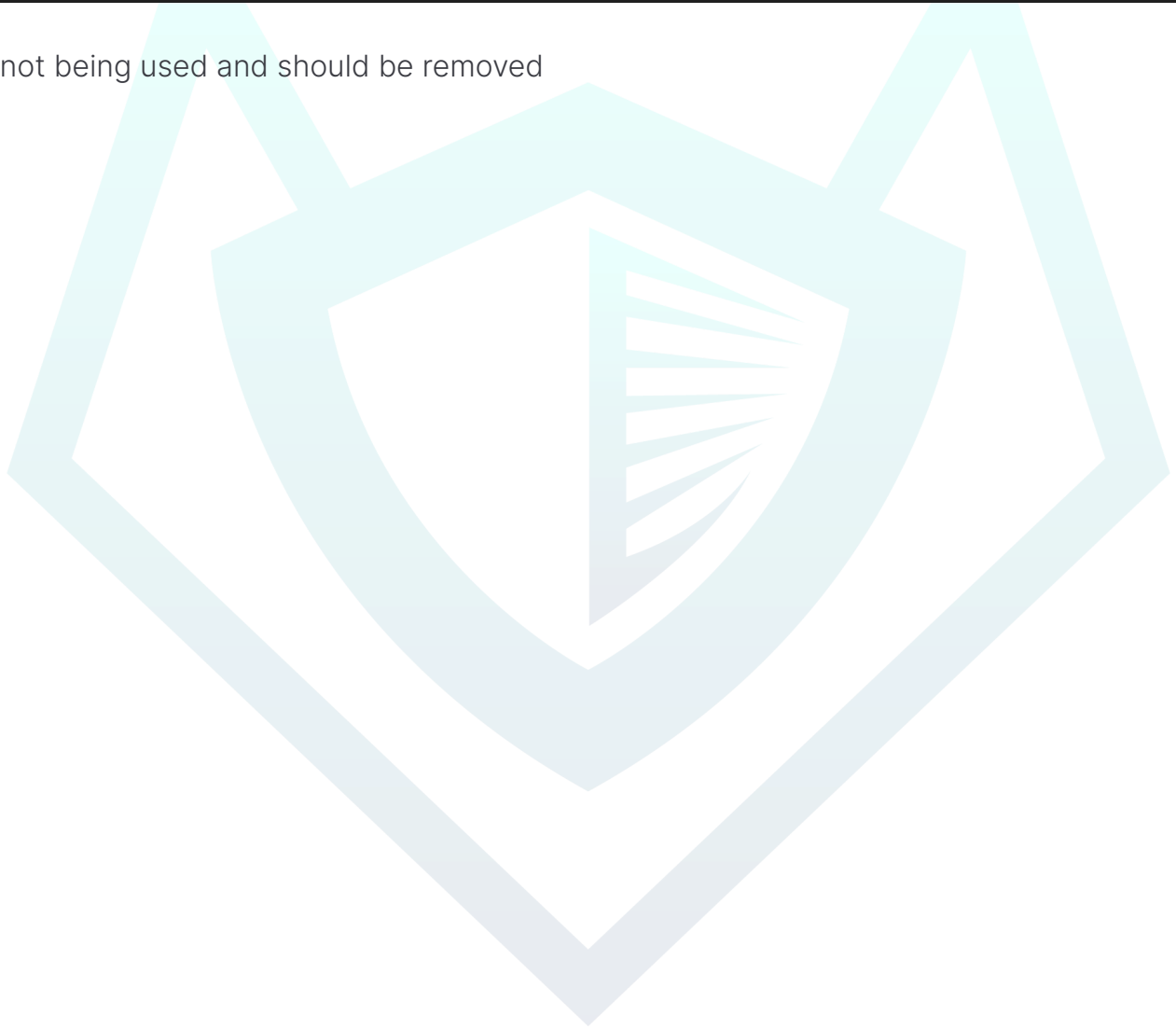
The codes above may result in an error or an unexpected behavior since stratCandidate.implementation is using both not equal (!=) and equal(=). Pick only one(1) condition for the logic.

SWC-135 | Code With No Effects

Functions

```
functionCall  
functionCallWithValue  
functionDelegateCall  
functionStaticCall  
sendValue
```

Is not being used and should be removed



SWC-102/103 / CW-001 | A Floating Pragma is Set / Multiple Versions across all Contracts

Code

```
// SPDX-License-Identifier: MIT
// File: @openzeppelin/contracts/utils/Context.sol

pragma solidity >=0.6.0 <0.8.0;
```

The compiler version should be a fixed one to avoid undiscovered compiler bugs. Fixed version sample below

```
// SPDX-License-Identifier: MIT
// File: @openzeppelin/contracts/utils/Context.sol

pragma solidity 0.6.12;
```


AUDIT COMMENTS | Purple Wave Finance

Smart Contract audit comment for a non-technical perspective

- Contract is not pausable
- Owner can renounce and transfer ownership
- Owner can burn tokens
- Owner cannot mint after initial deployment
- Owner cannot set max transaction limit
- Owner cannot block users
- Users can deposit & earn
- Users can withdraw
- Owner can take trapped/foreign tokens inside the contract





CONTRACTWOLF

Blockchain Security - Smart Contract Audits