Cyberscope

Audit Report

SuperWhale

March 2024

Network	BSC
Address	0xB0477d6A8c31E6082ea891D2F4e6a181A07Bb1F5
Audited by	© cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	RME	Repetitive Methods Execution	Unresolved
●	MTEE	Missing Transfer Event Emission	Unresolved
●	PMRM	Potential Mocked Router Manipulation	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	RED	Redudant Event Declaration	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	TUU	Time Units Usage	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved

●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Analysis **1**
Diagnostics **2**
Table of Contents **4**
Review **6**
Audit Updates 6
Source Files 7
Findings Breakdown **9**
ST - Stops Transactions 10
Description 10
Recommendation 10
RME - Repetitive Methods Execution 12
Description 12
Recommendation 12
MTEE - Missing Transfer Event Emission 13
Description 13
Recommendation 13
PMRM - Potential Mocked Router Manipulation 14
Description 14
Recommendation 15
PTRP - Potential Transfer Revert Propagation 17
Description 17
Recommendation 17
PVC - Price Volatility Concern 18
Description 18
Recommendation 18
RED - Redudant Event Declaration 19
Description 19
Recommendation 19
RSW - Redundant Storage Writes 20
Description 20
Recommendation 21
TUU - Time Units Usage 22
Description 22
Recommendation 22
L02 - State Variables could be Declared Constant 23
Description 23
Recommendation 23
L04 - Conformance to Solidity Naming Conventions 24
Description 24

Recommendation 25
L05 - Unused State Variable 26
Description 26
Recommendation 26
L07 - Missing Events Arithmetic 27
Description 27
Recommendation 27
L09 - Dead Code Elimination 28
Description 28
Recommendation 28
L13 - Divide before Multiply Operation 30
Description 30
Recommendation 30
L14 - Uninitialized Variables in Local Scope 31
Description 31
Recommendation 31
L16 - Validate Variable Setters 32
Description 32
Recommendation 32
L17 - Usage of Solidity Assembly 33
Description 33
Recommendation 33
L19 - Stable Compiler Version 34
Description 34
Recommendation 34
Functions Analysis **35**
Inheritance Graph **45**
Flow Graph **46**
Summary **47**
Disclaimer **48**
About Cyberscope **49**

Review

Contract Name	SPW
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	1 runs
Explorer	https://bscscan.com/address/0xb0477d6a8c31e6082ea891d2f4e6a181a07bb1f5
Address	0xb0477d6a8c31e6082ea891d2f4e6a181a07bb1f5
Network	BSC
Symbol	SPW
Decimals	9
Total Supply	100,000,000
Badge Eligibility	Yes

Audit Updates

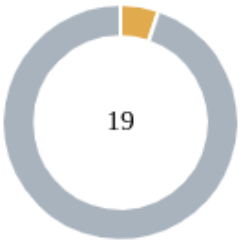
Initial Audit	09 Mar 2024 https://github.com/cyberscope-io/audits/blob/main/2-spw/v1/audit.pdf
Corrected Phase 2	11 Mar 2024 https://github.com/cyberscope-io/audits/blob/main/2-spw/v2/audit.pdf
Corrected Phase 3	12 Mar 2024

Source Files

Filename	SHA256
packages/lib/FeeType.sol	d5e3cf6baf0c242335bea6323c852e1b3c3f3c87a6b9292f29c0af3e841f9578
packages/lib/CurrencyTransferLib.sol	023a538c07fc30e8ab53c9c82ccadbb6a64be48337f58403f6682f706500a2f9
packages/infra/interface/IWETH.sol	839869bd411a4e68c9a59d2a0c394a087641eeeadeda4956a255dc3179110cc3
packages/infra/interface/IThirdwebContract.sol	8fc9d29ddee99b052ccdc521c272ee4df8a7de0e1754bfcba397dc5cdfa18c72
packages/external-deps/openzeppelin/token/ERC20/utils/SafeERC20.sol	3f614f5257a97f9a63fab94efb2d85c187fbb9cc964eda956719f2b646cfc636
packages/eip/interface/IERC20.sol	d16bec2c3503cabd12eb803b9b9d4fbf4ceaba3ac5541d2c705564abc1ed52a1
contracts/SPW.sol	f5158d7f8e2e14f314f39d13bac0826c0697df4e09ccedd74e2b8bd8bbab7dd4
@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol	e7b950eee23563e23989a3b51a1456614a1838084eef1fad04eb2be0bc280f48
@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol	5fb301961e45cb482fe4e05646d2f529aa449fe0e90c6671475d6a32356fa2d4
@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol	35fb271561f3dc72e91b3a42c6e40c2bb2e788cd8ca58014ac43f6198b8d32ca
@openzeppelin/contracts-upgradeable/utils/structs/EnumerableSetUpgradeable.sol	014846dc6c387e8fd6d31df636c28898eed601dc668725edbb1a0606c58d4b7e
@openzeppelin/contracts-upgradeable/utils/introspection/IERC165Upgradeable.sol	a39bc026ad6214e9ecd526bd4a1ddf9862d80bd4a9d0d031d9bafa4c3c147c0b

@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol	fd84e5284eccc479268f0ef36b830019d4f7999ceb7959430d8d8d9e602dd4ef
@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol	cd823c76cbf5f5b6ef1bda565d58be66c843c37707cd93eb8fb5425deebd6756
@openzeppelin/contracts-upgradeable/access/IAccessControlUpgradeable.sol	6d3fbd4566bc123db1ee6ba2a1b79544b572df9b9cc9be360ddb3244dd07c86b
@openzeppelin/contracts-upgradeable/access/IAccessControlEnumerableUpgradeable.sol	00e174801c04f08f2840ee1eed6394d06ba2b029c0b6078166255148794c1187
@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol	86752f503f326b20940831c24fd682d21767235c88dfd5960a43a17c148c93ed
@openzeppelin/contracts-upgradeable/access/AccessControlEnumerableUpgradeable.sol	a55a53b215e2bb9c350bf7b86ee09b0e488522f7d8747877fd9a3a7e474c2c26
@openzeppelin/contracts/utils/Address.sol	8160a4242e8a7d487d940814e5279d934e81f0436689132a4e73394bab084a6d

Findings Breakdown



Critical	0
Medium	1
Minor / Informative	18

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	1	0	0	0
Minor / Informative	18	0	0	0

ST - Stops Transactions

Criticality	Medium
Location	contracts/SPW.sol#L617,1005
Status	Unresolved

Description

The contract owner has the authority to stop the buys for all users by setting the `_transactionBuyLimit` to zero. As a result, the users will not be able to buy tokens.

```
// Ensure that amount is within the limit in case we are buying
if (isPancakeSwapPair(sender)) {
    require(
        amount <= _transactionBuyLimit,
        "Buy amount exceeds the maximum allowed"
    );
}

    function setTransactionBuyLimit(uint256 limit) public onlyOwner {
        _transactionBuyLimit = limit * 10 ** DECIMALS;
    }
```

Additionally, the contract owner has the authority to stop transactions, as described in detail in sections `PMRM` and `PTRP`. As a result, the contract might operate as a honeypot.

Recommendation

The contract could embody a check for not allowing setting the `_transactionBuyLimit` less than a reasonable amount. A suggested implementation could check that the minimum amount should be more than a fixed percentage of the total supply. The team should carefully manage the private keys of the owner's account. We

strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

RME - Repetitive Methods Execution

Criticality	Minor / Informative
Location	contracts/SPW.sol#L1294,1331
Status	Unresolved

Description

The contract executes the `isMarketTransfer(sender, recipient)` repetitively in the smart contract code. This method is used to determine whether a transfer operation involves market transactions. Repeating the invocation of `isMarketTransfer(sender, recipient)` for each transfer operation can lead to unnecessary gas consumption and decrease the efficiency of the contract. Instead, the contract can optimize its performance by executing the method once and caching the result for subsequent use within the same transaction.

```
if (!isMarketTransfer(sender, recipient)) {
    return;
}
```

Recommendation

To address this finding, it is recommended to refactor the contract code to execute `isMarketTransfer(sender, recipient)` once per transaction and store the result in a variable. Subsequent calls within the same transaction can then utilize the cached result rather than executing the method repeatedly. This optimization contributes to the overall optimization and gas efficiency of the smart contract.

MTEE - Missing Transfer Event Emission

Criticality	Minor / Informative
Location	contracts/SPW.sol#L701
Status	Unresolved

Description

The contract is a missing transfer event emission when fees are transferred to the contract address as part of the transfer process. This omission can lead to a lack of visibility into fee transactions and hinder the ability of decentralized applications (DApps) like blockchain explorers to accurately track and analyze these transactions.

```
// Add fees to contract
_balances[address(this)] += feeAmount;
```

Recommendation

To address this issue, it is recommended to emit a transfer event after transferring the taxed amount to the contract address. The event should include relevant information such as the sender, recipient (contract address), and the amount transferred.

PMRM - Potential Mocked Router Manipulation

Criticality	Minor / Informative
Location	contracts/SPW.sol#L933
Status	Unresolved

Description

The contract includes a method that allows the owner to modify the router address and create a new pair. While this feature provides flexibility, it introduces a security threat. The owner could set the router address to any contract that implements the router's interface, potentially containing malicious code. In the event of a transaction triggering the swap functionality with such a malicious contract as the router, the transaction may be manipulated.

```
   function setPancakeSwapRouter(address routerAddress) public
onlyOwner {
        require(
            routerAddress != address(0),
            "Cannot use the zero address as router address"
        );

        _pancakeSwapRouterAddress = routerAddress;
        _pancakeswapV2Router =
IPancakeRouter02(_pancakeSwapRouterAddress);

        address pancakePair;

        // REC Certik : Confirm pair does not already exist
        address get_pair =
IPancakeFactory(_pancakeswapV2Router.factory())
            .getPair(address(this),
_pancakeswapV2Router.WETH());

        if (get_pair == address(0)) {
            pancakePair =
IPancakeFactory(_pancakeswapV2Router.factory())
                .createPair(address(this),
_pancakeswapV2Router.WETH());
        } else {
            pancakePair = get_pair;
        }

        _pancakeswapV2Pair = pancakePair;

        onPancakeSwapRouterUpdated();
    }
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.

- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	contracts/SPW.sol#L793,803,811
Status	Unresolved

Description

The contract sends funds to a `marketingWallet` as part of the transfer flow. This address can either be a wallet address or a contract. If the address belongs to a contract then it may revert from incoming payment. As a result, the error will propagate to the token's contract and revert the transfer.

```
        (, uint bnbAddedToLiquidity, ) =
_pancakeswapV2Router.addLiquidityETH{
            value: bnbToBeAddedToLiquidity
        }(
            ...
            _autoLiquidityWallet,
            ...
        );
...
        (bool sent, ) = _marketingWallet.call{value:
bnbToBeSendToMarketing}(
            ""
        );
        require(sent, "Failed to send BNB to marketing
wallet");
...
        (sent, ) = _devAddress.call{value:
bnbToBeSendToDev}("");
        require(sent, "Failed to send BNB to dev wallet");
```

Recommendation

The contract should tolerate the potential revert from the underlying contracts when the interaction is part of the main transfer flow. This could be achieved by not allowing set contract addresses or by sending the funds in a non-revertable way.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	contracts/SPW.sol#L746
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `_tokenSwapThreshold` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract will swap a huge amount of tokens for ETH.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

```
        uint256 tokensAvailableForSwap =
balanceOf(address(this));
         if (tokensAvailableForSwap >= _tokenSwapThreshold) {
             // Limit to threshold
             tokensAvailableForSwap = _tokenSwapThreshold;

             // Make sure that we are not stuck in a loop (Swap
only once)
             bool isSelling = isPancakeSwapPair(recipient);
             if (isSelling) {
                 executeSwap(tokensAvailableForSwap);
             }
```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the exchange reserves. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

RED - Redudant Event Declaration

Criticality	Minor / Informative
Location	contracts/SPW.sol#L520
Status	Unresolved

Description

The contract uses events that are not emitted within the contract's functions. As a result, these declared events are redundant and serve no purpose within the contract's current implementation.

```
event AutoBurned(uint256 bnbAmount);
```

Recommendation

To optimize contract performance and efficiency, it is advisable to regularly review and refactor the codebase, removing the unused event declarations. This proactive approach not only streamlines the contract, reducing deployment and execution costs but also enhances readability and maintainability.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	contracts/SPW.sol#L1114
Status	Unresolved

Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
    function setSwapEnabled(bool isEnabled) public onlyOwner {
        _isSwapEnabled = isEnabled;
    }


    function setFeeEnabled(bool isEnabled) public onlyOwner {
        _isFeeEnabled = isEnabled;
    }

    function setTokenHoldEnabled(bool isEnabled) public
onlyOwner {
        _isTokenHoldEnabled = isEnabled;
    }

    function setExcludedFromFees(address addr, bool value)
public onlyOwner {
        _addressesExcludedFromFees[addr] = value;
    }

    function setExcludedFromHold(address addr, bool value)
public onlyOwner {
        _addressesExcludedFromHold[addr] = value;
    }

    function activateBuying(bool isEnabled) public onlyOwner {
        _isBuyingAllowed = isEnabled;
    }
```

Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

TUU - Time Units Usage

Criticality	Minor / Informative
Location	contracts/SPW.sol#L1207,1732
Status	Unresolved

Description

The contract is using arbitrary numbers to form time-related values. As a result, it decreases the readability of the codebase and prevents the compiler to optimize the source code.

```
   uint256 private _rewardCyclePeriod = 43200; // The duration
of the reward cycle (e.g. can claim rewards once 12 hours)

    function setRewardCyclePeriod(uint256 period) public
onlyOwner {
        require(
            period >= 3600 && period <= 86400,
            "RewardCycle must be updated to between 1 and 24
hours"
        );
        _rewardCyclePeriod = period;
    }
```

Recommendation

It is a good practice to use the time units reserved keywords like `seconds` , `minutes` , `hours` , `days` and `weeks` to process time-related calculations.

It's important to note that these time units are simply a shorthand notation for representing time in seconds, and do not have any effect on the actual passage of time or the execution of the contract. The time units are simply a convenience for expressing time in a more human-readable form.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/SPW.sol#L1179,1182,1185
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
uint128 private platformFeeBps
address internal platformFeeRecipient
address public primarySaleRecipient
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/SPW.sol#L182,186,217,230,466,478,505,928,1202,1231,1232,1234,1919,1924
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
function __ReentrancyGuard_init() internal onlyInitializing {
        __ReentrancyGuard_init_unchained();
    }

function __ReentrancyGuard_init_unchained() internal
onlyInitializing {
        _status = _NOT_ENTERED;
...
function WETH() external pure returns (address);
uint256 private constant _totalTokens = 1 * 10 ** 8 * 10 **
DECIMALS
uint8 public _transferFee
mapping(address => bool) public _whiteList
address _add
bool _val
string calldata _uri

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

Find more information on the Solidity documentation https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	contracts/SPW.sol#L1164,1169,1176,1179,1182,1188
Status	Unresolved

Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
bytes32 private constant TYPEHASH =
        keccak256(
            "MintRequest(address to,address
primarySaleRecipient,uint256 quantity,uint256 price,address
currency,uint128 validityStartTimestamp,uint128
validityEndTimestamp,bytes32 uid)"
        )
bytes32 internal constant MINTER_ROLE =
keccak256("MINTER_ROLE")
uint128 internal constant MAX_BPS = 10_000
uint128 private platformFeeBps
address internal platformFeeRecipient
mapping(bytes32 => bool) private minted
```

Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	contracts/SPW.sol#L1738,1744,1759,1767,1776,1808,1837,1846,1866,1908,1925
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
_rewardCyclePeriod = period
_rewardCycleExtensionThreshold = threshold
_maxClaimAllowed = value
_minRewardBalance = balance
_maxGasForAutoClaim = gas
_globalRewardDampeningPercentage = value
_gradualBurnMagnitude = magnitude
_gradualBurnTimespan = timespan
_mainBnbPoolSize = size
_sendWeiGasLimit = amount
antiBlockNum = _blockNum
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/SPW.sol#L60,182,186
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function _msgDataCheck() internal view virtual returns (bytes
calldata) {
        this; // silence state mutability warning without
generating bytecode - see
https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }

function __ReentrancyGuard_init() internal onlyInitializing {
        __ReentrancyGuard_init_unchained();
    }

function __ReentrancyGuard_init_unchained() internal
onlyInitializing {
        _status = _NOT_ENTERED;
    }
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	contracts/SPW.sol#L766,768,775,785,802,810,1575,1580,1620,1629
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
uint256 tokensToSwapForLiquidity = tokensReservedForLiquidity /
2
uint256 bnbToBeAddedToLiquidity = (bnbSwapped *
            tokensToSwapForLiquidity) / tokensToSwap
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	contracts/SPW.sol#L14560
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
bool sent
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/SPW.sol#L956
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
_pancakeswapV2Pair = pancakePair
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	contracts/SPW.sol#L1697
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {
        size := extcodesize(account)
      }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/SPW.sol#L11
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.11;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
Context	Implementation			
	_msgSenderCheck	Internal		
	_msgDataCheck	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
ReentrancyGuardUpgradeable	Implementation	Initializable		
	__ReentrancyGuard_init	Internal	✓	onlyInitializing

	__ReentrancyGuard_init_unchained	Internal	✓	onlyInitializing
IPancakeRouter01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-

IPancakeRouter02	Interface	IPancakeRouter01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
IPancakeFactory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
SPWBase	Implementation	Context, IERC20Metadata, Ownable, ReentrancyGuardUpgradeable, AccessControlEnumerableUpgradeable		

		Public	✓	-
	activate	Public	✓	onlyOwner
	onActivated	Internal	✓	
	balanceOf	Public		-
	transfer	Public	✓	-
	transferFrom	Public	✓	-
	approve	Public	✓	-
	doTransfer	Internal	✓	
	onBeforeTransfer	Internal	✓	
	onTransfer	Internal	✓	
	updateBalances	Private	✓	
	doApprove	Private	✓	
	calculateFeeRate	Private		
	onBeforeCalculateFeeRate	Internal		
	executeSwapIfNeeded	Private	✓	
	executeSwap	Private	✓	
	swapTokensForBNB	Internal	✓	
	swapBNBForTokens	Internal	✓	
	isSellTransferLimited	Private		
	isSwapTransfer	Private		
	isMarketTransfer	Internal		
	amountUntilSwap	Public		-
	increaseAllowance	Public	✓	-

	decreaseAllowance	Public	✓	-
	addToWhiteList	Public	✓	onlyOwner
	setPancakeSwapRouter	Public	✓	onlyOwner
	onPancakeSwapRouterUpdated	Internal	✓	
	isPancakeSwapPair	Internal		
	setFees	Public	✓	onlyOwner
	setTransferFee	Public	✓	onlyOwner
	setTransactionSellLimit	Public	✓	onlyOwner
	transactionSellLimit	Public		-
	setTransactionBuyLimit	Public	✓	onlyOwner
	transactionBuyLimit	Public		-
	setHoldLimit	Public	✓	onlyOwner
	holdLimit	Public		-
	setTokenSwapThreshold	Public	✓	onlyOwner
	tokenSwapThreshold	Public		-
	name	Public		-
	symbol	Public		-
	totalSupply	Public		-
	decimals	Public		-
	allowance	Public		-
	pancakeSwapRouterAddress	Public		-
	pancakeSwapPairAddress	Public		-
	autoLiquidityWallet	Public		-

	setAutoLiquidityWallet	Public	✓	onlyOwner
	marketingWallet	Public		-
	setMarketingWallet	Public	✓	onlyOwner
	devWallet	Public		-
	setDevWallet	Public	✓	onlyOwner
	totalFeesPooled	Public		-
	totalBNBLiquidityAddedFromFees	Public		-
	isSwapEnabled	Public		-
	setSwapEnabled	Public	✓	onlyOwner
	isFeeEnabled	Public		-
	setFeeEnabled	Public	✓	onlyOwner
	isTokenHoldEnabled	Public		-
	setTokenHoldEnabled	Public	✓	onlyOwner
	isExcludedFromFees	Public		-
	setExcludedFromFees	Public	✓	onlyOwner
	isExcludedFromHold	Public		-
	setExcludedFromHold	Public	✓	onlyOwner
	activateBuying	Public	✓	onlyOwner
		External	Payable	-
SPW	Implementation	SPWBase		
	contractType	External		-
	contractVersion	External		-

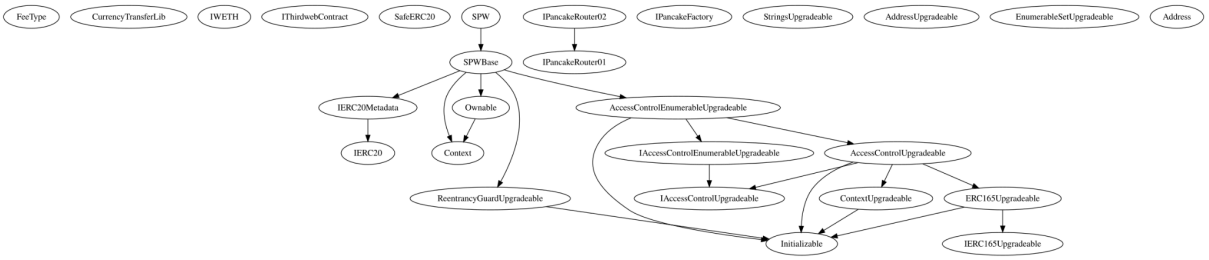
	setContractURI	External	✓	onlyRole
		Public	✓	SPWBase
	onActivated	Internal	✓	
	onBeforeTransfer	Internal	✓	
	onTransfer	Internal	✓	
	processGradualBurn	Private	✓	
	updateAutoClaimQueue	Private	✓	
	claimReward	External	✓	isHuman nonReentrant
	claimReward	Public	✓	-
	doClaimReward	Private	✓	
	claimBNB	Private	✓	
	claimSPW	Private	✓	
	processRewardClaimQueue	Public	✓	-
	processRewardClaimQueueAndRefundGas	External	✓	-
	isRewardReady	Public		-
	isIncludedInRewards	Public		-
	calculateRewardCycleExtension	Public		-
	calculateClaimRewards	Public		-
	calculateBNBReward	Public		-
	onPancakeSwapRouterUpdated	Internal	✓	
	isMarketTransfer	Internal		
	isBurnTransfer	Private		
	shouldBurn	Public		-

	buyAndBurn	External	✓	onlyOwner
	doBuyAndBurn	Private	✓	
	isContract	Public		-
	totalAmountOfTokensHeld	Public		-
	bnbRewardClaimed	Public		-
	bnbRewardClaimedAsSPW	Public		-
	totalBNBClaimed	Public		-
	totalBNBClaimedAsSPW	Public		-
	rewardCyclePeriod	Public		-
	setRewardCyclePeriod	Public	✓	onlyOwner
	setRewardCycleExtensionThreshold	Public	✓	onlyOwner
	nextAvailableClaimDate	Public		-
	maxClaimAllowed	Public		-
	setMaxClaimAllowed	Public	✓	onlyOwner
	minRewardBalance	Public		-
	setMinRewardBalance	Public	✓	onlyOwner
	maxGasForAutoClaim	Public		-
	setMaxGasForAutoClaim	Public	✓	onlyOwner
	isAutoClaimEnabled	Public		-
	setAutoClaimEnabled	Public	✓	onlyOwner
	isExcludedFromRewards	Public		-
	setExcludedFromRewards	Public	✓	onlyOwner
	globalRewardDampeningPercentage	Public		-

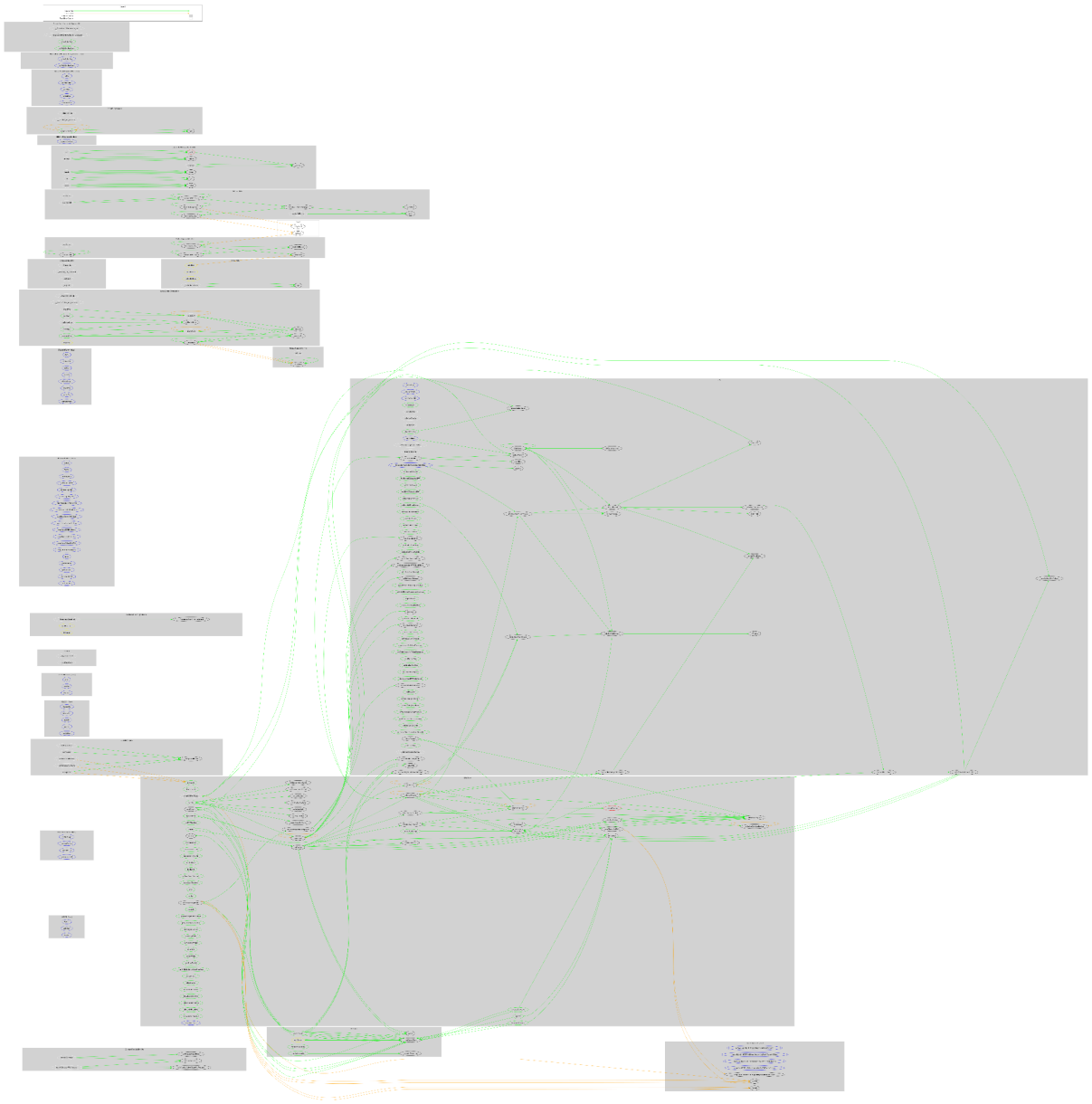
	setGlobalRewardDampeningPercentage	Public	✓	onlyOwner
	approveClaim	Public	✓	-
	isClaimApproved	Public		-
	isRewardAsTokensEnabled	Public		-
	setRewardAsTokensEnabled	Public	✓	onlyOwner
	gradualBurnMagnitude	Public		-
	setGradualBurnMagnitude	Public	✓	onlyOwner
	gradualBurnTimespan	Public		-
	setGradualBurnTimespan	Public	✓	onlyOwner
	claimRewardAsTokensPercentage	Public		-
	setClaimRewardAsTokensPercentage	Public	✓	-
	mainBnbPoolSize	Public		-
	setMainBnbPoolSize	Public	✓	onlyOwner
	isInRewardClaimQueue	Public		-
	reimburseAfterSPWClaimFailure	Public		-
	setReimburseAfterSPWClaimFailure	Public	✓	onlyOwner
	lastBurnDate	Public		-
	rewardClaimQueueLength	Public		-
	rewardClaimQueueIndex	Public		-
	isWhitelistedExternalProcessor	Public		-
	setWhitelistedExternalProcessor	Public	✓	onlyOwner
	setSendWeiGasLimit	Public	✓	onlyOwner
	setExcludeNonHumansFromRewards	Public	✓	onlyOwner

	disableAntiBot	Public	✓	onlyOwner
	updateAntiBotStatus	Private	✓	
	updateBlockNum	Public	✓	onlyOwner
	onBeforeCalculateFeeRate	Internal		

Inheritance Graph



Flow Graph



Summary

SuperWhale contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner like stop the buy transactions. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io