



Cyberscope

Audit Report

PigTWAP

July 2023

Network BSC

Address 0xda4e53f4a44aa05eab468a0559953c69d75960fd

Audited by © cyberscope

Table of Contents

Table of Contents	1
Review	3
Audit Updates	3
Source Files	3
Overview	4
Findings Breakdown	5
Diagnostics	6
RM - Redundant Multiplication	7
Description	7
Recommendation	7
Team Update	7
CR - Code Repetition	8
Description	8
Recommendation	8
DPI - Decimals Precision Inconsistency	9
Description	9
Recommendation	9
Team Update	10
IDI - Immutable Declaration Improvement	11
Description	11
Recommendation	11
L04 - Conformance to Solidity Naming Conventions	12
Description	12
Recommendation	13
Team Update	13
L09 - Dead Code Elimination	14
Description	14
Recommendation	14
Team Update	15
L13 - Divide before Multiply Operation	17
Description	17
Recommendation	17
Team Update	17
L14 - Uninitialized Variables in Local Scope	18
Description	18
Recommendation	18
Team Update	18
L16 - Validate Variable Setters	19
Description	19

Recommendation	19
Team Update	19
Functions Analysis	20
Inheritance Graph	24
Flow Graph	25
Summary	26
Disclaimer	27
About Cyberscope	28

Review

Explorer	https://bscscan.com/address/0xda4e53f4a44aa05eab468a0559953c69d75960fd
----------	---

Audit Updates

Initial Audit	08 Jul 2023
Corrected Phase 2	09 Jul 2023

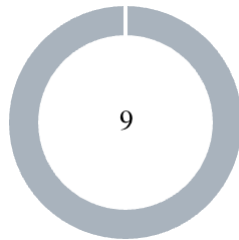
Source Files

Filename	SHA256
TWAP.sol	39c42fba94a8e6eef8730aeecc6ed0bf64b2a64f9b6216289e9e98b9feb9da2c

Overview

The PigTWAP contract provides a way to calculate the average price of a token pair on Uniswap over time, which can be useful for various applications such as automated trading, price monitoring, and liquidity provision.

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	9

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	9	0	0	0

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	RM	Redundant Multiplication	Acknowledged
●	CR	Code Repetition	Acknowledged
●	DPI	Decimals Precision Inconsistency	Acknowledged
●	IDI	Immutable Declaration Improvement	Acknowledged
●	L04	Conformance to Solidity Naming Conventions	Acknowledged
●	L09	Dead Code Elimination	Acknowledged
●	L13	Divide before Multiply Operation	Acknowledged
●	L14	Uninitialized Variables in Local Scope	Acknowledged
●	L16	Validate Variable Setters	Acknowledged

RM - Redundant Multiplication

Criticality	Minor / Informative
Location	contracts/PigTWAP.sol#L558
Status	Acknowledged

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

Multiplication by 1 is unnecessary because it does not change the value of the number being multiplied. When you multiply a number by 1, the result is always the same as the original number. Therefore, it has no effect on the final outcome or computation.

```
uint multiplier = 1*10**18;
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

Team Update

multiplication value must be present for inherited FixedPoint uq112x112 data type (from uniswap V2 oracle library) to work.

CR - Code Repetition

Criticality	Minor / Informative
Location	contracts/PigTWAP.sol#L592,595
Status	Acknowledged

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

```
price0Average = FixedPoint.uq112x112(  
    uint224((price0Cumulative - price0CumulativeLast) / timeElapsed)  
);  
price1Average = FixedPoint.uq112x112(  
    uint224((price1Cumulative - price1CumulativeLast) / timeElapsed)  
);
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

DPI - Decimals Precision Inconsistency

Criticality	Minor / Informative
Location	contracts/PigTWAP.sol#L558,611,613
Status	Acknowledged

Description

The decimals field of a contract's ERC20 token can be used to specify the number of decimal places that the token uses. For example, if decimals are set to `8`, it means that the smallest unit of the token is `0.00000001`, and if decimals are set to `18`, it means that the smallest unit of the token is `0.000000000000000001`.

However, there is an inconsistency in the way that the decimals field is handled in some ERC20 contracts. The ERC20 specification does not specify how the decimals field should be implemented, and as a result, some contracts use different precision numbers.

This inconsistency can cause problems when interacting with these contracts, as it is not always clear how the decimals field should be interpreted. For example, if a contract expects the decimals field to be 18 digits, but the contract being interacted with uses 8 digits, the result of the interaction may not be what was expected.

```
uint multiplier = 1*10**18; // TWAP per single token
..
amountOut = price0Average.mul(multiplier).decode144();
```

Recommendation

To avoid these issues, it is important to carefully review the implementation of the decimals field of the underlying tokens. The team is advised to normalize each decimal to one single source of truth. A recommended way is to scale all the decimals to the greatest token's decimal. Hence, the contract will not lose precision in the calculations.

The following example depicts 3 tokens with different decimals precision.

ERC20	Decimals
-------	----------

Token 1	6
Token 2	9
Token 3	18

All the decimals could be normalized to 18 since it represents the ERC20 token with the greatest digits.

Team Update

Function designed only for ProfitPig token which is 18 decimals. Not designed for widespread decimal-independent usage.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	TWAP.sol#L534
Status	Acknowledged

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
auth
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	TWAP.sol#L214,220,367,368,385,525
Status	Acknowledged

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.

```
struct uq112x112 {
    uint224 _x;
}

struct uq144x112 {
    uint256 _x;
}

function DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint);
uint constant multiplier = 1*10**18
```

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>.

Team Update

Inherited code from existing audited contract, not authored by us. Not implementing due to backward compatibility risk and regression test requirement etc.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	TWAP.sol#L27,64,113,230,235,240,259,267,295,331,339,409,416,427,434,441,451,460,471
Status	Acknowledged

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 mostSignificantBit(uint256 x) internal pure returns (uint8 r) {
    require(x > 0, 'BitMath::mostSignificantBit: zero');

    if (x >= 0x100000000000000000000000000000000) {
        x >>= 128;
        r += 128;
        ...
    }
    if (x >= 0x4) {
        x >>= 2;
        r += 2;
    }
    if (x >= 0x2) r += 1;
}

...
```

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.

Team Update

Inherited code from existing audited contract, not authored by us. Not implementing due to backward compatibility risk and regression test requirement etc.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	TWAP.sol#L176,177,180,181,182,183,184,185,186,187,188
Status	Acknowledged

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 precision.

```
d /= pow2  
r *= 2 - d * r
```

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.

Team Update

Inherited code from existing audited contract, not authored by us. Not implementing due to backward compatibility risk and regression test requirement etc.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	TWAP.sol#L464
Status	Acknowledged

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.

```
uint i
```

Recommendation

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

Team Update

Inherited code from existing audited contract, not authored by us. Not implementing due to backward compatibility risk and regression test requirement etc.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	TWAP.sol#L534
Status	Acknowledged

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.

```
auth = _auth
```

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.

Team Update

Not implementing as these values will only be set once during construction by deployer/owner.

Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
SafeMath	Library			
	add	Internal		
	sub	Internal		
	mul	Internal		
BitMath	Library			
	mostSignificantBit	Internal		
	leastSignificantBit	Internal		
Babylonian	Library			
	sqrt	Internal		
FullMath	Library			
	fullMul	Internal		
	fullDiv	Private		
	mulDiv	Internal		
FixedPoint	Library			

	encode	Internal		
	encode144	Internal		
	decode	Internal		
	decode144	Internal		
	mul	Internal		
	muli	Internal		
	muluq	Internal		
	divuq	Internal		
	fraction	Internal		
	reciprocal	Internal		
	sqrt	Internal		
IUniswapV2Pair	Interface			
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-

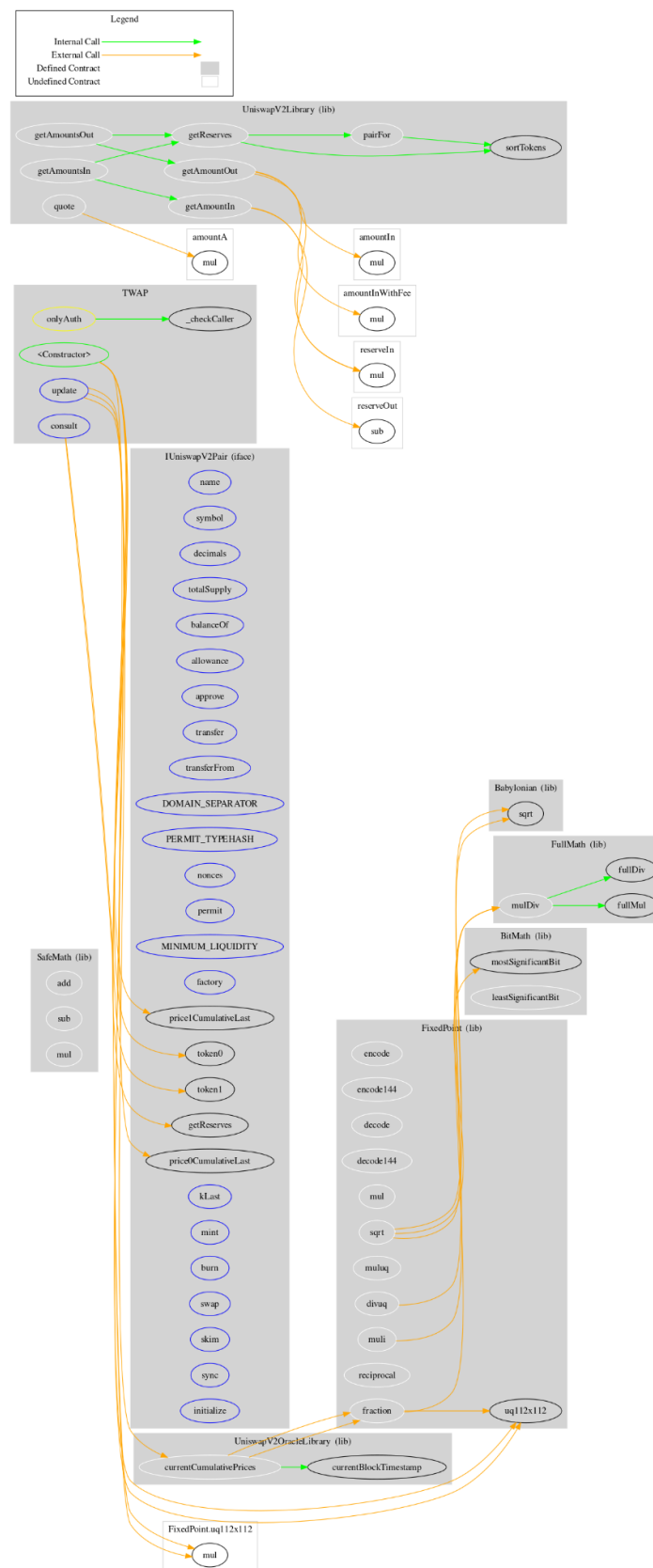
	PERMIT_TYPEHASH	External		-
	nonces	External		-
	permit	External	✓	-
	MINIMUM_LIQUIDITY	External		-
	factory	External		-
	token0	External		-
	token1	External		-
	getReserves	External		-
	price0CumulativeLast	External		-
	price1CumulativeLast	External		-
	kLast	External		-
	mint	External	✓	-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
UniswapV2Library	Library			
	sortTokens	Internal		
	pairFor	Internal		
	getReserves	Internal		
	quote	Internal		

	getAmountOut	Internal		
	getAmountIn	Internal		
	getAmountsOut	Internal		
	getAmountsIn	Internal		
UniswapV2OracleLibrary	Library			
	currentBlockTimestamp	Internal		
	currentCumulativePrices	Internal		
TWAP	Implementation			
		Public	✓	-
	_checkCaller	Internal		
	update	External	✓	onlyAuth
	consult	External		-

Inheritance Graph



Flow Graph



Summary

ProfitPIG contract implements a utility mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

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>



Cyberscope

Audit Report

ProfitPIG

July 2023

Network BSC

Address 0x806674e94fbd7e50752ea0e97b2041fa72d5d000

Audited by © cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	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
●	FSA	Fixed Swap Address	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Analysis	1
Diagnostics	2
Table of Contents	3
Review	5
Audit Updates	5
Source Files	5
Findings Breakdown	6
FSA - Fixed Swap Address	7
Description	7
Recommendation	7
Team Update	7
RSW - Redundant Storage Writes	8
Description	8
Recommendation	8
Team Update	8
RSML - Redundant SafeMath Library	9
Description	9
Recommendation	9
Team Update	9
IDI - Immutable Declaration Improvement	10
Description	10
Recommendation	10
L04 - Conformance to Solidity Naming Conventions	11
Description	11
Recommendation	12
L07 - Missing Events Arithmetic	13
Description	13
Recommendation	13
L09 - Dead Code Elimination	14
Description	14
Recommendation	14
Team Update	15
L16 - Validate Variable Setters	16
Description	16
Recommendation	16
L17 - Usage of Solidity Assembly	17
Description	17
Recommendation	17
Team Update	17

L20 - Succeeded Transfer Check	18
Description	18
Recommendation	18
Team Update	18
Functions Analysis	19
Inheritance Graph	26
Flow Graph	27
Summary	28
Disclaimer	29
About Cyberscope	30

Review

Contract Name	ProfitPIG
Compiler Version	v0.8.0+commit.c7dfd78e
Optimization	200 runs
Explorer	https://bscscan.com/address/0x806674e94fbd7e50752ea0e97b2041fa72d5d000
Address	0x806674e94fbd7e50752ea0e97b2041fa72d5d000
Network	BSC
Symbol	PIG
Decimals	18
Total Supply	100,000,000

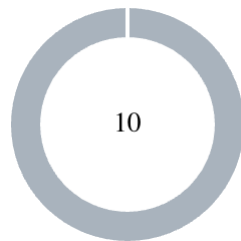
Audit Updates

Initial Audit	07 Jul 2023
Corrected Phase 2	09 Jul 2023

Source Files

Filename	SHA256
ProfitPIG.sol	252d6c0c518b941f0fd95732292cebccff08f008568bebf5a3060acd63633a11

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	10

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	10	0	0	0

FSA - Fixed Swap Address

Criticality	Minor / Informative
Location	contracts/ProfitPig.sol#L1076,1077
Status	Acknowledged

Description

The swap address is assigned once and it can not be changed. It is a common practice in decentralized exchanges to create new swap versions. A contract that cannot change the swap address may not be able to catch up to the upgrade. As a result, the contract will not be able to migrate to a new liquidity pool pair or decentralized exchange.

```
router = IUniswapV2Router02(router_);  
pair = IUniswapV2Factory(router.factory()).createPair(  
    address(this),  
    router.WETH()  
);
```

Recommendation

The team is advised to add the ability to change the pair and router address in order to cover potential liquidity pool migrations. It would be better to support multiple pair addresses so the token will be able to have the same behavior in all the decentralized liquidity pairs.

Team Update

Target Router is not an upgradeable proxy:

<https://bscscan.com/address/0x10ed43c718714eb63d5aa57b78b54704e256024e#code>

and most DEX's like Uniswap and Pancakeswap are releasing new versions (e.g. V3) instead of modifying V2. Very unlikely requirement.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	contracts/ProfitPig.sol#L1104,1108,1369
Status	Acknowledged

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract modifies the state of certain variables event when their current state matches the provided argument. As a result, the contract performs redundant storage writes.

```
TWAP = _twap;  
feesActive = true;  
isFeeExempt[holder] = true;
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

Team Update

Functions present to modify state only, current state will not match provided arguments

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	ProfitPIG.sol
Status	Acknowledged

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on

<https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes>.

Team Update

Requires re-work and regression test simply for gas savings which team are not allocating time to.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	ProfitPIG.sol#L1076,1077,1078,1080,1081,1082,1087,1089,1090
Status	Acknowledged

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
_name  
_symbol  
_totalSupply  
rewardToken  
router  
pair  
distributor  
distributorGas  
swapThreshold
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	ProfitPIG.sol#L472,796,847,848,1015,1016,1029,1030,1031,1032,1033,1037,1108
Status	Acknowledged

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 WETH() external pure returns (address);
address public _token
uint256 _minPeriod
uint256 _minDistribution
address public TWAP
uint8 private constant _decimals = 18
uint256 public constant liquidityFee = 4
uint256 public constant buybackFee = 4
uint256 public constant reflectionFee = 4
uint256 public constant totalFee = 12
uint256 public constant feeDenominator = 100
uint256 public constant minBuybackThreshold = 2 * 10 ** 18
address _twap
```

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>.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	ProfitPIG.sol#L850
Status	Acknowledged

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.

```
minPeriod = _minPeriod
```

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	ProfitPIG.sol#L247,275,300,310,329,343,362,372,389,399,416,1327
Status	Acknowledged

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 isContract(address account) internal view returns (bool) {
    // This method relies on extcodesize, which returns 0 for contracts in
    // construction, since the code is only stored at the end of the
    // constructor execution.

    uint256 size;
    assembly {
        size := extcodesize(account)
    }
    return size > 0;
}

...
```

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.

Team Update

Inherited code from existing audited contract, not authored by us. Not implementing due to backward compatibility risk and regression test requirement etc.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	ProfitPIG.sol#L768,1080,1110
Status	Acknowledged

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.

```
owner = adr  
rewardToken = rewardToken_  
TWAP = _twap
```

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	ProfitPIG.sol#L253,428
Status	Acknowledged

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)  
}  
  
assembly {  
    let returndata_size := mload(returndata)  
    revert(add(32, returndata), returndata_size)  
}
```

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.

Team Update

Inherited code from existing audited contract, not authored by us. Not implementing due to backward compatibility risk and regression test requirement etc.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	ProfitPIG.sol#L943
Status	Acknowledged

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
rewardToken.transfer(shareholder, amount)
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the [Openzeppelin library](#).

Team Update

Inherited code from existing audited contract, not authored by us. Not implementing due to backward compatibility risk and regression test requirement etc.

Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
SafeMath	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	sub	Internal		
	div	Internal		
	mod	Internal		
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	

	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResult	Internal		
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
IUniswapV2Router01	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		-
IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-

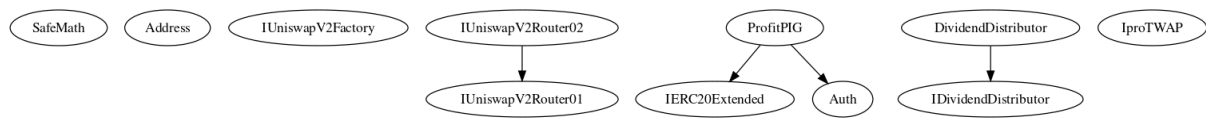
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
IERC20Extended	Interface			
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Auth	Implementation			
		Public	✓	-
	authorize	Public	✓	onlyOwner
	unauthorize	Public	✓	onlyOwner
	isOwner	Public		-
	isAuthorized	Public		-
	transferOwnership	Public	✓	onlyOwner
	renounceOwnership	Public	✓	onlyOwner
IDividendDistributor	Interface			

	setDistributionCriteria	External	✓	-
	setShare	External	✓	-
	deposit	External	Payable	-
	process	External	✓	-
DividendDistributor	Implementation	IDividendDistributor		
		Public	✓	-
	setDistributionCriteria	External	✓	onlyToken
	setShare	External	✓	onlyToken
	deposit	External	Payable	onlyToken
	process	External	✓	onlyToken
	shouldDistribute	Internal		
	distributeDividend	Internal	✓	
	claimDividend	External	✓	-
	getUnpaidEarnings	Public		-
	getCumulativeDividends	Internal		
	addShareholder	Internal	✓	
	removeShareholder	Internal	✓	
IproTWAP	Interface			
	update	External	✓	-
	consult	External		-

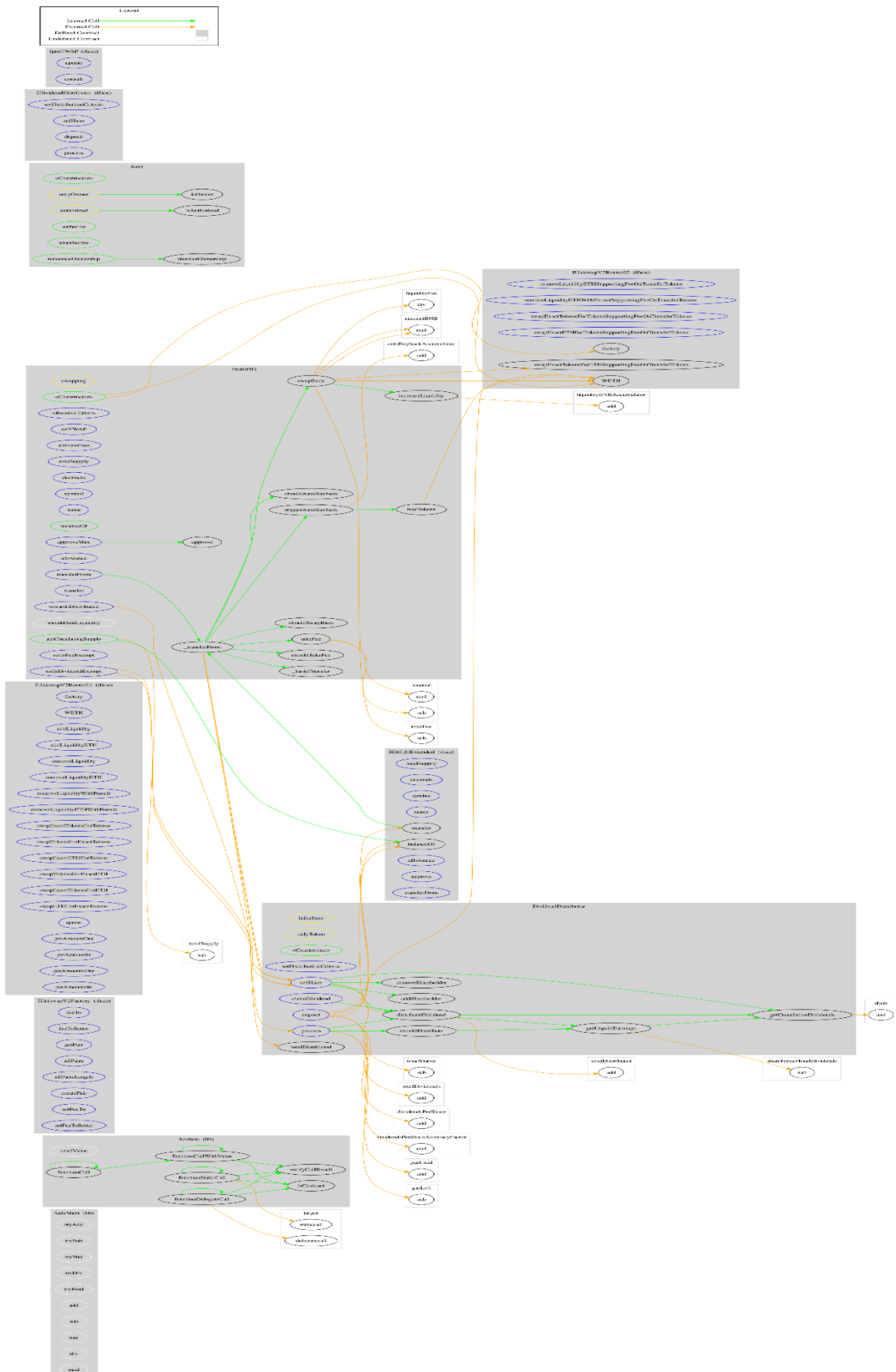
ProfitPIG	Implementation	IERC20Extended, Auth		
		Public	Payable	Auth
		External	Payable	-
	setTWAP	External	✓	onlyOwner
	activateFees	External	✓	onlyOwner
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	balanceOf	Public		-
	rewardsDistributed	External		-
	allowance	External		-
	approve	Public	✓	-
	approveMax	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	_transferFrom	Internal	✓	
	_basicTransfer	Internal	✓	
	shouldTakeFee	Internal		
	takeFee	Internal	✓	
	shouldSwapBack	Internal		
	swapBack	Internal	✓	swapping
	shouldAddLiquidity	Internal		

	increaseLiquidity	Internal	✓	
	shouldAutoBuyback	Internal		
	triggerAutoBuyback	Internal	✓	
	buyTokens	Internal	✓	swapping
	setIsDividendExempt	External	✓	authorized
	setIsFeeExempt	External	✓	authorized
	getCirculatingSupply	Public		-

Inheritance Graph



Flow Graph



Summary

ProfitPIG contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. ProfitPIG is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler errors. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions. There is also a fixed amount of 12% sell fees.

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>



Cyberscope

Audit Report

PigTWAP

July 2023

SHA256 cbdee9ed3f049fce188a4c5177e31f96f6936e684c44d8a34f105f7f29f69715

Audited by © cyberscope

Table of Contents

Table of Contents	1
Review	3
Audit Updates	3
Source Files	3
Overview	4
Findings Breakdown	5
Diagnostics	6
DPI - Decimals Precision Inconsistency	7
Description	7
Recommendation	8
CR - Code Repetition	9
Description	9
Recommendation	9
RM - Redundant Multiplication	10
Description	10
Recommendation	10
IDI - Immutable Declaration Improvement	11
Description	11
Recommendation	11
L02 - State Variables could be Declared Constant	12
Description	12
Recommendation	12
L04 - Conformance to Solidity Naming Conventions	13
Description	13
Recommendation	14
L09 - Dead Code Elimination	15
Description	15
Recommendation	16
L13 - Divide before Multiply Operation	17
Description	17
Recommendation	17
L14 - Uninitialized Variables in Local Scope	18
Description	18
Recommendation	18
L16 - Validate Variable Setters	19
Description	19
Recommendation	19
L18 - Multiple Pragma Directives	20
Description	20

Recommendation	20
Functions Analysis	21
Inheritance Graph	25
Flow Graph	26
Summary	27
Disclaimer	28
About Cyberscope	29

Review

Testing Deploy	https://testnet.bscscan.com/address/0x1f7bd6fd16d905883f4638654803bae43bb27eb3
----------------	---

Audit Updates

Initial Audit	08 Jul 2023
---------------	-------------

Source Files

Filename	SHA256
contracts/PigTWAP.sol	cbdee9ed3f049fce188a4c5177e31f96f6936e684c44d8a34f105f7f29f69715

Overview

The PigTWAP contract provides a way to calculate the average price of a token pair on Uniswap over time, which can be useful for various applications such as automated trading, price monitoring, and liquidity provision.

Findings Breakdown



● Critical	0
● Medium	0
● Minor / Informative	11

Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	0	0	0	0
● Medium	0	0	0	0
● Minor / Informative	11	0	0	0

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	DPI	Decimals Precision Inconsistency	Unresolved
●	CR	Code Repetition	Unresolved
●	RM	Redundant Multiplication	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L18	Multiple Pragma Directives	Unresolved

DPI - Decimals Precision Inconsistency

Criticality	Minor / Informative
Location	contracts/PigTWAP.sol#L558,611,613
Status	Unresolved

Description

The decimals field of a contract's ERC20 token can be used to specify the number of decimal places that the token uses. For example, if decimals are set to `8`, it means that the smallest unit of the token is `0.00000001`, and if decimals are set to `18`, it means that the smallest unit of the token is `0.00000000000000000001`.

However, there is an inconsistency in the way that the decimals field is handled in some ERC20 contracts. The ERC20 specification does not specify how the decimals field should be implemented, and as a result, some contracts use different precision numbers.

This inconsistency can cause problems when interacting with these contracts, as it is not always clear how the decimals field should be interpreted. For example, if a contract expects the decimals field to be 18 digits, but the contract being interacted with uses 8 digits, the result of the interaction may not be what was expected.

```
uint multiplier = 1*10**18; // TWAP per single token
..
amountOut = price0Average.mul(multiplier).decode144();
```

Recommendation

To avoid these issues, it is important to carefully review the implementation of the decimals field of the underlying tokens. The team is advised to normalize each decimal to one single source of truth. A recommended way is to scale all the decimals to the greatest token's decimal. Hence, the contract will not lose precision in the calculations.

The following example depicts 3 tokens with different decimals precision.

ERC20	Decimals
Token 1	6
Token 2	9
Token 3	18

All the decimals could be normalized to 18 since it represents the ERC20 token with the greatest digits.

CR - Code Repetition

Criticality	Minor / Informative
Location	contracts/PigTWAP.sol#L592,595
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

```
price0Average = FixedPoint.uq112x112(  
    uint224((price0Cumulative - price0CumulativeLast) / timeElapsed)  
);  
price1Average = FixedPoint.uq112x112(  
    uint224((price1Cumulative - price1CumulativeLast) / timeElapsed)  
);
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

RM - Redundant Multiplication

Criticality	Minor / Informative
Location	contracts/PigTWAP.sol#L558
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

Multiplication by 1 is unnecessary because it does not change the value of the number being multiplied. When you multiply a number by 1, the result is always the same as the original number. Therefore, it has no effect on the final outcome or computation.

```
uint multiplier = 1*10**18;
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/PigTWAP.sol#L567
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
auth
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/PigTWAP.sol#L558
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.

```
uint multiplier = 1*10**18
```

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/PigTWAP.sol#L232,238,387,388,405
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.

```
struct uq112x112 {  
    uint224 _x;  
}  
  
struct uq144x112 {  
    uint256 _x;  
}  
  
function DOMAIN_SEPARATOR() external view returns (bytes32);  
function PERMIT_TYPEHASH() external pure returns (bytes32);  
function MINIMUM_LIQUIDITY() external pure returns (uint);
```

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>.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/PigTWAP.sol#L30,67,121,248,253,258,277,285,313,349,357,433,440,451,458,465,475,484,495
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 mostSignificantBit(uint256 x) internal pure returns (uint8 r) {
    require(x > 0, 'BitMath::mostSignificantBit: zero');

    if (x >= 0x100000000000000000000000000000000) {
        x >>= 128;
        r += 128;
    }
    ...
    if (x >= 0x4) {
        x >>= 2;
        r += 2;
    }
    if (x >= 0x2) r += 1;
}
...
```

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/PigTWAP.sol#L187,188,191,192,193,194,195,196,197,198,199
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 precision.

```
d /= pow2  
r *= 2 - d * r
```

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/PigTWAP.sol#L488
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.

```
uint i
```

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/PigTWAP.sol#L567
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.

```
auth = _auth
```

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.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	contracts/PigTWAP.sol#L4,25,114,169,223,370,425,508,544
Status	Unresolved

Description

If the contract includes multiple conflicting pragma directives, it may produce unexpected errors. To avoid this, it's important to include the correct pragma directive at the top of the contract and to ensure that it is the only pragma directive included in the contract.

```
pragma solidity =0.6.6;  
pragma solidity >=0.5.0;  
pragma solidity >=0.4.0;  
pragma solidity 0.6.6;
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

By including all required compiler options and flags in a single pragma directive, the potential conflicts could be avoided and ensure that the contract can be compiled correctly.

Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
SafeMath	Library			
	add	Internal		
	sub	Internal		
	mul	Internal		
BitMath	Library			
	mostSignificantBit	Internal		
	leastSignificantBit	Internal		
Babylonian	Library			
	sqrt	Internal		
FullMath	Library			
	fullMul	Internal		
	fullDiv	Private		
	mulDiv	Internal		
FixedPoint	Library			

	encode	Internal		
	encode144	Internal		
	decode	Internal		
	decode144	Internal		
	mul	Internal		
	muli	Internal		
	muluq	Internal		
	divuq	Internal		
	fraction	Internal		
	reciprocal	Internal		
	sqrt	Internal		
IUniswapV2Pair	Interface			
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-

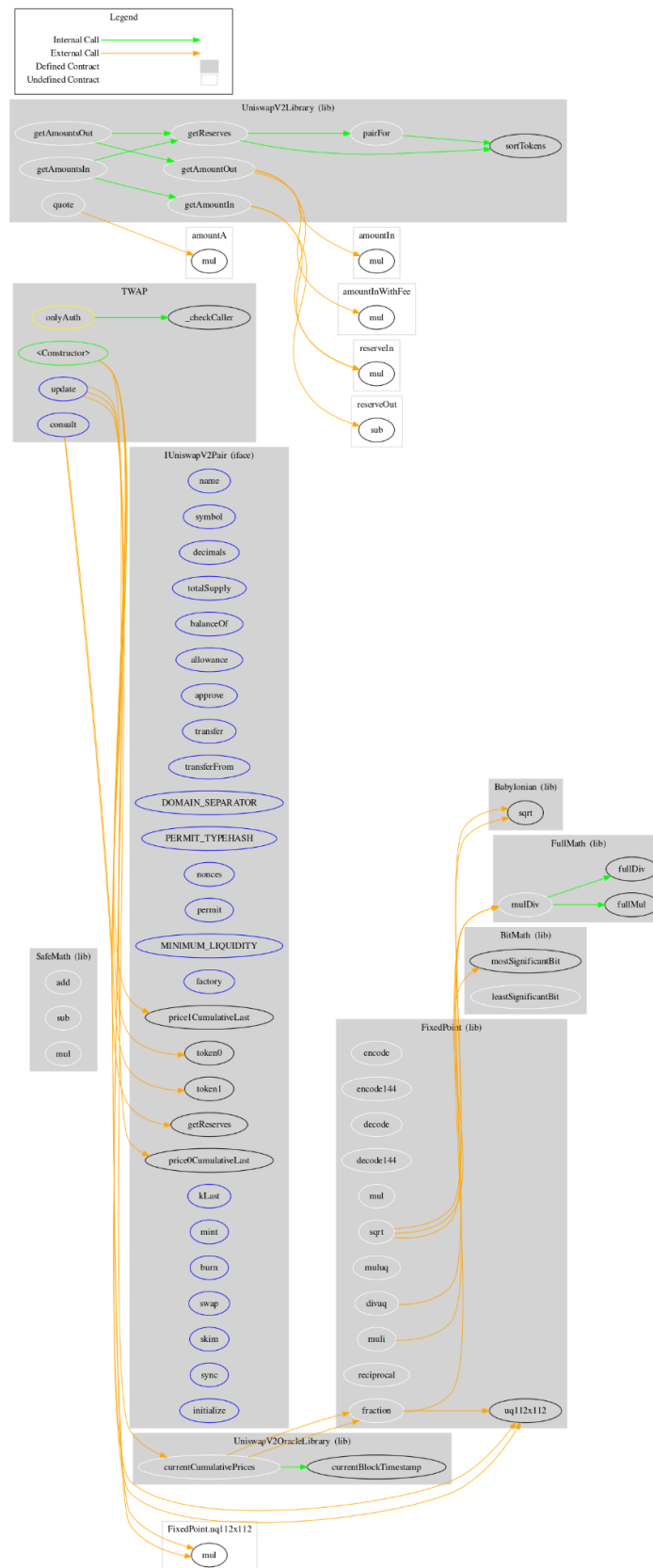
	PERMIT_TYPEHASH	External		-
	nonces	External		-
	permit	External	✓	-
	MINIMUM_LIQUIDITY	External		-
	factory	External		-
	token0	External		-
	token1	External		-
	getReserves	External		-
	price0CumulativeLast	External		-
	price1CumulativeLast	External		-
	kLast	External		-
	mint	External	✓	-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
UniswapV2Library	Library			
	sortTokens	Internal		
	pairFor	Internal		
	getReserves	Internal		
	quote	Internal		

	getAmountOut	Internal		
	getAmountIn	Internal		
	getAmountsOut	Internal		
	getAmountsIn	Internal		
UniswapV2OracleLibrary	Library			
	currentBlockTimestamp	Internal		
	currentCumulativePrices	Internal		
TWAP	Implementation			
		Public	✓	-
	_checkCaller	Internal		
	update	External	✓	onlyAuth
	consult	External		-

Inheritance Graph



Flow Graph



Summary

ProfitPIG contract implements a utility mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

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>



Cyberscope

Audit Report

ProfitPIG

July 2023

SHA256 88d445f3dd4116dfff476d56775c9bdf4b1ab296554c36d8160662c0de0acc20

Audited by © cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	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
●	US	Untrusted Source	Unresolved
●	FSA	Fixed Swap Address	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Analysis	1
Diagnostics	2
Table of Contents	3
Review	5
Audit Updates	5
Source Files	5
Findings Breakdown	6
US - Untrusted Source	7
Description	7
Recommendation	7
FSA - Fixed Swap Address	8
Description	8
Recommendation	8
RSW - Redundant Storage Writes	9
Description	9
Recommendation	9
MEE - Missing Events Emission	10
Description	10
Recommendation	10
RSML - Redundant SafeMath Library	11
Description	11
Recommendation	11
IDI - Immutable Declaration Improvement	12
Description	12
Recommendation	12
L02 - State Variables could be Declared Constant	13
Description	13
Recommendation	13
L04 - Conformance to Solidity Naming Conventions	14
Description	14
Recommendation	15
L07 - Missing Events Arithmetic	16
Description	16
Recommendation	16
L09 - Dead Code Elimination	17
Description	17
Recommendation	18
L16 - Validate Variable Setters	19
Description	19

Recommendation	19
L17 - Usage of Solidity Assembly	20
Description	20
Recommendation	20
L19 - Stable Compiler Version	21
Description	21
Recommendation	21
L20 - Succeeded Transfer Check	22
Description	22
Recommendation	22
Functions Analysis	23
Inheritance Graph	30
Flow Graph	31
Summary	32
Disclaimer	33
About Cyberscope	34

Review

Contract Name	ProfitPIG
Testing Deploy	https://testnet.bscscan.com/address/0x35f5fe16f2e49b08a953c12ee44d0799bcfb9d1a
Symbol	PPIG
Decimals	18
Total Supply	1,000,000

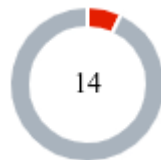
Audit Updates

Initial Audit	07 Jul 2023
---------------	-------------

Source Files

Filename	SHA256
contracts/ProfitPig.sol	88d445f3dd4116dfff476d56775c9bdf4b1ab296554c36d8160662c0de0acc20

Findings Breakdown



● Critical	1
● Medium	0
● Minor / Informative	13

Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	1	0	0	0
● Medium	0	0	0	0
● Minor / Informative	13	0	0	0

US - Untrusted Source

Criticality	Critical
Location	contracts/ProfitPig.sol#L1103
Status	Unresolved

Description

The contract uses an external contract in order to determine the transaction's flow. The external contract is untrusted. As a result, it may produce security issues and harm the transactions.

```
function setTWAP(address _twap) external onlyOwner {  
    TWAP = _twap;  
}
```

Recommendation

The contract should use a trusted external source. A trusted source could be either a commonly recognized or an audited contract. The pointing addresses should not be able to change after the initialization.

FSA - Fixed Swap Address

Criticality	Minor / Informative
Location	contracts/ProfitPig.sol#L1076,1077
Status	Unresolved

Description

The swap address is assigned once and it can not be changed. It is a common practice in decentralized exchanges to create new swap versions. A contract that cannot change the swap address may not be able to catch up to the upgrade. As a result, the contract will not be able to migrate to a new liquidity pool pair or decentralized exchange.

```
router = IUniswapV2Router02(router_);  
pair = IUniswapV2Factory(router.factory()).createPair(  
    address(this),  
    router.WETH()  
);
```

Recommendation

The team is advised to add the ability to change the pair and router address in order to cover potential liquidity pool migrations. It would be better to support multiple pair addresses so the token will be able to have the same behavior in all the decentralized liquidity pairs.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	contracts/ProfitPig.sol#L1104,1108,1369
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract modifies the state of certain variables event when their current state matches the provided argument. As a result, the contract performs redundant storage writes.

```
TWAP = _twap;  
feesActive = true;  
isFeeExempt[holder] = true;
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	contracts/ProfitPig.sol#L1104,1108,1369
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
TWAP = _twap;  
feesActive = true;  
isFeeExempt[holder] = true;
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/ProfitPig.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on

<https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes>.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/ProfitPig.sol#L1071,1072,1073,1075,1076,1077,1082,1084,1085
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
_name  
_symbol  
_totalSupply  
rewardToken  
router  
pair  
distributor  
distributorGas  
swapThreshold
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/ProfitPig.sol#L1028,1029,1030,1031,1032,1035
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.

```
uint256 public liquidityFee = 4
uint256 public buybackFee = 4
uint256 public reflectionFee = 4
uint256 public totalFee = 12
uint256 public feeDenominator = 100
uint256 public minBuybackThreshold = 2 * 10 ** 18
```

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/ProfitPig.sol#L472,796,847,848,1015,1016,1103
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 WETH() external pure returns (address);
address public _token
uint256 _minPeriod
uint256 _minDistribution
address public TWAP
uint8 private constant _decimals = 18
address _twap
```

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>.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	contracts/ProfitPig.sol#L850
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.

```
minPeriod = _minPeriod
```

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/ProfitPig.sol#L247,275,300,310,329,343,362,372,389,399,416,1314
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 isContract(address account) internal view returns (bool) {
    // This method relies on extcodesize, which returns 0 for
    contracts in
    // construction, since the code is only stored at the end of the
    // constructor execution.

    uint256 size;
    assembly {
        size := extcodesize(account)
    }
    return size > 0;
}

...
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/ProfitPig.sol#L768,1075,1104
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.

```
owner = adr  
rewardToken = rewardToken_  
TWAP = _twap
```

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/ProfitPig.sol#L253,428
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)  
}  
  
assembly {  
    let returndata_size := mload(returndata)  
    revert(add(32, returndata), returndata_size)  
}
```

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/ProfitPig.sol#L2
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 ^0.8.0;
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	contracts/ProfitPig.sol#L943
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
rewardToken.transfer(shareholder, amount)
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the [Openzeppelin library](#).

Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
SafeMath	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	sub	Internal		
	div	Internal		
	mod	Internal		
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	

	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResult	Internal		
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
IUniswapV2Router01	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		-
IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-

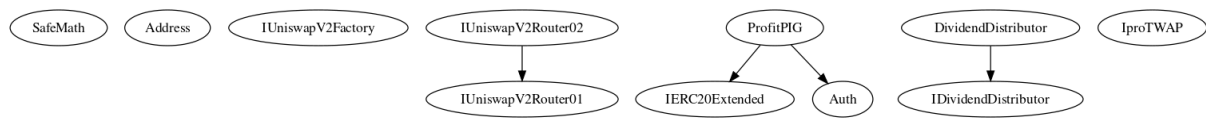
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
IERC20Extended	Interface			
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Auth	Implementation			
		Public	✓	-
	authorize	Public	✓	onlyOwner
	unauthorize	Public	✓	onlyOwner
	isOwner	Public		-
	isAuthorized	Public		-
	transferOwnership	Public	✓	onlyOwner
	renounceOwnership	Public	✓	onlyOwner

IDividendDistributor	Interface			
	setDistributionCriteria	External	✓	-
	setShare	External	✓	-
	deposit	External	Payable	-
	process	External	✓	-
DividendDistributor	Implementation	IDividendDistributor		
		Public	✓	-
	setDistributionCriteria	External	✓	onlyToken
	setShare	External	✓	onlyToken
	deposit	External	Payable	onlyToken
	process	External	✓	onlyToken
	shouldDistribute	Internal		
	distributeDividend	Internal	✓	
	claimDividend	External	✓	-
	getUnpaidEarnings	Public		-
	getCumulativeDividends	Internal		
	addShareholder	Internal	✓	
	removeShareholder	Internal	✓	
IproTWAP	Interface			
	update	External	✓	-
	consult	External		-

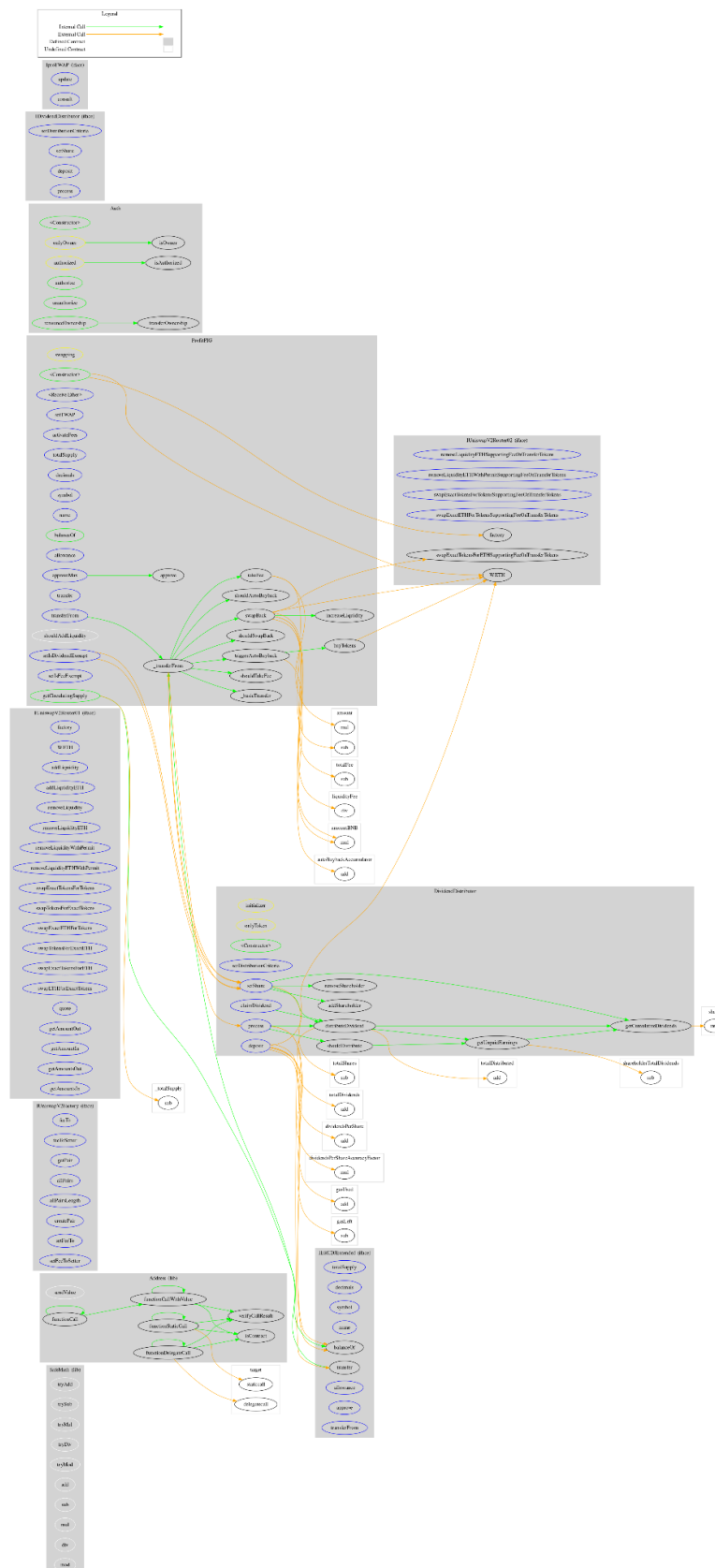
ProfitPIG	Implementation	IERC20Extended, Auth		
		Public	Payable	Auth
		External	Payable	-
	setTWAP	External	✓	onlyOwner
	activateFees	External	✓	onlyOwner
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	balanceOf	Public		-
	allowance	External		-
	approve	Public	✓	-
	approveMax	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	_transferFrom	Internal	✓	
	_basicTransfer	Internal	✓	
	shouldTakeFee	Internal		
	takeFee	Internal	✓	
	shouldSwapBack	Internal		
	swapBack	Internal	✓	swapping
	shouldAddLiquidity	Internal		

	increaseLiquidity	Internal	✓	
	shouldAutoBuyback	Internal		
	triggerAutoBuyback	Internal	✓	
	buyTokens	Internal	✓	swapping
	setIsDividendExempt	External	✓	authorized
	setIsFeeExempt	External	✓	authorized
	getCirculatingSupply	Public		-

Inheritance Graph



Flow Graph



Summary

ProfitPIG contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. ProfitPIG is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler errors. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions. There is also a fixed amount of 12% sell fees.

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>