Cyberscope

Audit Report

Payout Coin

June 2023

Network GOERLI ETH

Address 0xBCdB9622d9b885ac2537666358AC2a4E45875143

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	Unresolved
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Critical	ZD	Zero Division	Unresolved
Minor / Informative	PLE	Potential Locked Ether	Unresolved
Minor / Informative	PTRP	Potential Transfer Revert Propagation	Unresolved
Minor / Informative	RV	Randomization Vulnerability	Unresolved
Minor / Informative	PVC	Price Volatility Concern	Unresolved
Minor / Informative	IDI	Immutable Declaration Improvement	Unresolved
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor / Informative	L07	Missing Events Arithmetic	Unresolved
Minor / Informative	L09	Dead Code Elimination	Unresolved
Minor / Informative	L16	Validate Variable Setters	Unresolved
Minor / Informative	L17	Usage of Solidity Assembly	Unresolved

Table of Contents

Review

Contract Name	Pay
Compiler Version	v0.8.7+commit.e28d00a7
Optimization	200 runs
Explorer	https://goerli.etherscan.io/address/0xbcdb9622d9b885ac2537666358ac2a4e45875143
Address	0xbcdb9622d9b885ac2537666358ac2a4e45875143
Network	GOERLI
Symbol	Pay
Decimals	9
Total Supply	1.000.000.000

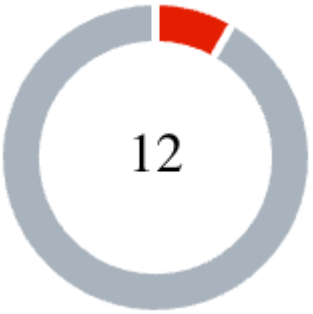
Audit Updates

Initial Audit	28 May 2023 https://github.com/cyberscope-io/audits/blob/main/2-pay/v1/audit.pdf
Corrected Phase 2	03 Jun 2023 https://github.com/cyberscope-io/audits/blob/main/2-pay/v2/audit.pdf
Corrected Phase 3	09 Jun 2023

Source Files

Filename	SHA256
Pay.sol	83e3ecef09c885e80e928a52295998d35069b3fdbe14e2d7895543e8273cf3c9

Findings Breakdown



Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	0	0	0	0
Minor / Informative	11	0	0	0

ELFM - Exceeds Fees Limit

Criticality	Minor / Informative
Location	Pay.sol#L455
Status	Unresolved

Description

The contract owner has the authority to increase over the allowed limit of 25%. The owner may take advantage of it by calling the fee setter functions with a high 10. As a result, the total fees will sum up to 30%.

```
function setMarketingFeePercent(uint256 updatedMarketingFee) external
onlyOwner {
  require(updatedMarketingFee <= 10, "Fee is crossing the boundaries");
  marketingFee = updatedMarketingFee;
  totalFee = liquidityFee + marketingFee + lotteryFee;

}

function setLotteryFeePercent(uint256 updatedLotteryFee) external onlyOwner
{
  require(updatedLotteryFee <= 10, "Fee is crossing the boundaries");
  lotteryFee = updatedLotteryFee;
  totalFee = liquidityFee + marketingFee + lotteryFee;

}


function setLiquidityFeePercent(uint256 updatedLiquidityFee) external
onlyOwner {
  require(liquidityFee <= 10, "Fee is crossing the boundaries");
  liquidityFee = updatedLiquidityFee;
  totalFee = liquidityFee + marketingFee + lotteryFee;

}
```

Recommendation

The contract could embody a check for the maximum acceptable value. 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. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

ZD - Zero Division

Criticality	Critical
Location	Pay.sol#L581
Status	Unresolved

Description

The contract is using variables that may be set to zero as denominators. This can lead to unpredictable and potentially harmful results, such as a transaction revert.

The variable `totalFee` coud be set to zero.

```
function swapAndLiquify(uint256 tokensToLiquify) private lockTheSwap {
    uint256 tokensToLP = (tokensToLiquify * liquidityFee / totalFee) / 2 ;
    ...
}
```

Recommendation

It is important to handle division by zero appropriately in the code to avoid unintended behavior and to ensure the reliability and safety of the contract. The contract should ensure that the divisor is always non-zero before performing a division operation. It should prevent the variables to be set to zero, or should not allow the execution of the corresponding statements.

PLE - Potential Locked Ether

Criticality	Minor / Informative
Location	Pay.sol#L579
Status	Unresolved

Description

During the swap method, the contract calculates the swapped ETH by subtracting the `ethForLottery` . If the contract receives ETH from an external address, these ETH are not calculated by the swap method. As a result, they will be locked in the contract forever.

```
uint256 ethBalance = address(this).balance - ethForLottery;
uint256 ethFeeFactor = totalFee - (liquidityFee / 2);

uint256 ethForLiquidity = ethBalance * liquidityFee / ethFeeFactor /2 ;
uint256 ethForMarketing = (ethBalance * marketingFee) / (ethFeeFactor);
ethForLottery += ethBalance - (ethForLiquidity + ethForMarketing);
```

Recommendation

The team could calculate the swapped ETH by the difference between the before and after the `swapExactTokensForETHSupportingFeeOnTransferTokens()` . The remaining contract's ETH could be used for the lottery awards. That way the `ethForLottery` state variable could also be removed.

A suggested approach could be:

```
uint256 ethBalanceBefore = address(this).balance;
uniswapV2Router.swap..
uint256 ethBalance = address(this).balance - ethForLottery;
..
if(address(this).balance >= lotteryReward) {...}
```

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	Pay.sol#L614
Status	Unresolved

Description

The contract sends funds to a `marketingAddress` 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.

```
payable(marketingAddress).transfer(ethForMarketing);
```

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.

RV - Randomization Vulnerability

Criticality	Minor / Informative
Location	Pay.sol#L622
Status	Unresolved

Description

The contract is using an on-chain technique in order to determine random numbers. The blockchain runtime environment is fully deterministic, as a result, the pseudo-random numbers could be predicted.

```
    function random(uint number) public view returns(uint){
        return
uint(keccak256(abi.encodePacked(block.timestamp,block.difficulty,
        msg.sender))) % number;
    }
```

Recommendation

The contract could use an advanced randomization technique that guarantees an acceptable randomization factor. For instance, the Chainlink VRF (Verifiable Random Function). https://docs.chain.link/docs/chainlink-vrf

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	Pay.sol#L546
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `numTokensSellToAddToLiquidity` 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.

```
bool overMinTokenBalance = contractTokenBalance >=
numTokensSellToAddToLiquidity;
```

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 total supply. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	Pay.sol#L367,373,377
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.

```
_totalSupply
uniswapV2Pair
uniswapV2Router
```

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	Pay.sol#L135,136,153,175,324,325,326,327,331,478,483
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 DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint);
function WETH() external pure returns (address);
mapping (address => uint) internal _balances
mapping (address => mapping (address => uint)) internal _allowances
mapping (address => bool) public _isExcludedFromFee
mapping (address => bool) public AMMs
uint256 internal _totalSupply
bool _enabled
uint256 _numTokensSellToAddToLiquidity
```

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	Pay.sol#L453,461,468,485,507
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.

```
marketingFee = updatedMarketingFee
totalFee = liquidityFee + marketingFee + lotteryFee
liquidityFee = updatedLiquidityFee
numTokensSellToAddToLiquidity = _numTokensSellToAddToLiquidity
lotteryReward = amount
```

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	Pay.sol#L624
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 swapTokensForEth(uint256 tokenAmount) private {
        // generate the uniswap pair path of token -> weth
        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = uniswapV2Router.WETH();

...
            tokenAmount,
            0, // accept any amount of ETH
            path,
            address(this),
            block.timestamp
        );
    }
```

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	Pay.sol#L475
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.

```
marketingAddress = wallet
```

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	Pay.sol#L313
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 { codehash := extcodehash(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.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Context	Implementation			
		Public	✓	-
	_msgSender	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
ERC20Detailed	Implementation			

		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
IUniswapV2Pair	Interface			
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-

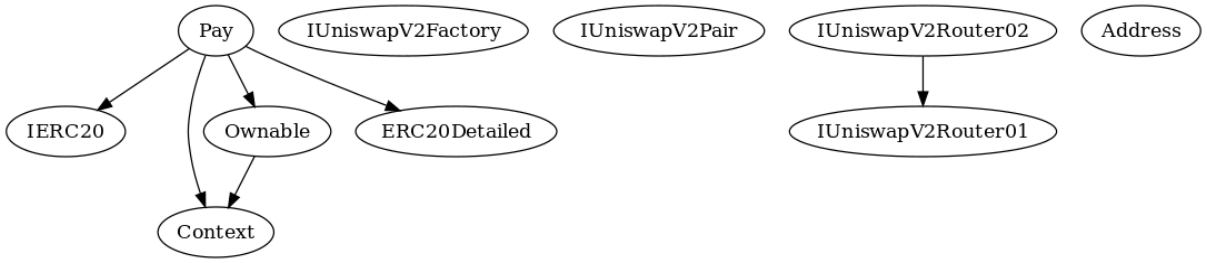
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-
	PERMIT_TYPEHASH	External		-
	nonces	External		-
	permit	External	✓	-
	MINIMUM_LIQUIDITY	External		-
	factory	External		-
	token0	External		-
	token1	External		-
	getReserves	External		-
	price0CumulativeLast	External		-
	price1CumulativeLast	External		-
	kLast	External		-
	mint	External	✓	-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
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	✓	-

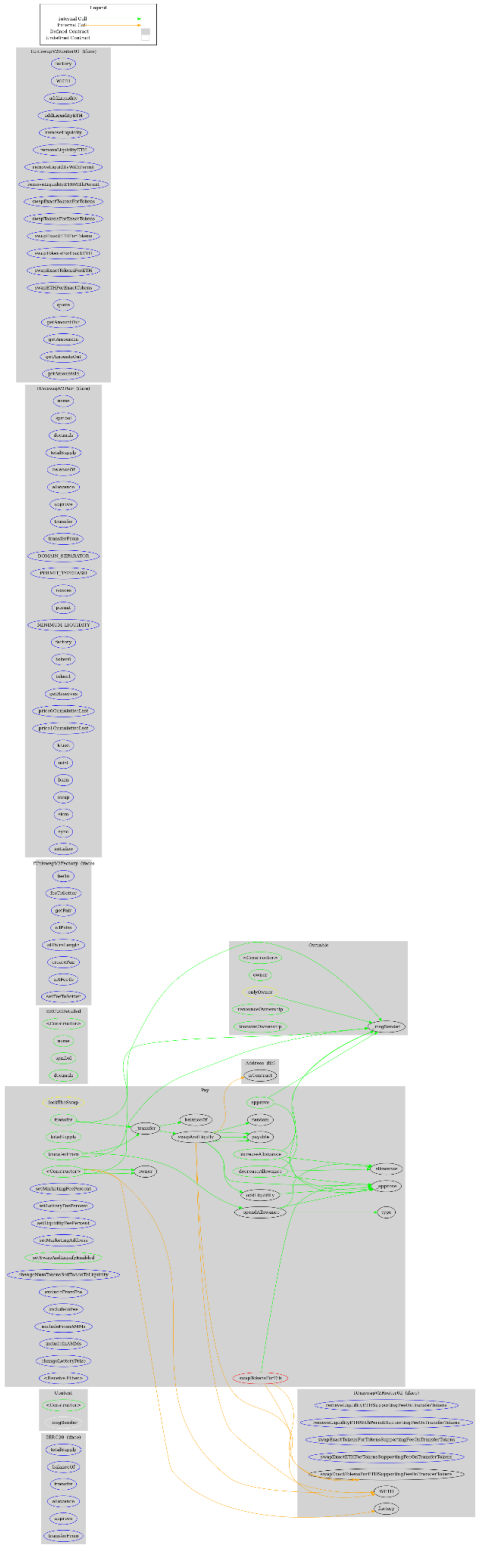
Address	Library			
	isContract	Internal		
Pay	Implementation	Context, Ownable, IERC20, ERC20Detailed		
		Public	✓	ERC20Detailed
	totalSupply	Public		-
	balanceOf	Public		-
	allowance	Public		-
	approve	Public	✓	-
	transfer	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_spendAllowance	Internal	✓	
	setMarketingFeePercent	External	✓	onlyOwner
	setLotteryFeePercent	External	✓	onlyOwner
	setLiquidityFeePercent	External	✓	onlyOwner
	setMarketingAddress	External	✓	onlyOwner
	setSwapAndLiquifyEnabled	Public	✓	onlyOwner
	changeNumTokensSellToAddToLiquidity	External	✓	onlyOwner
	excludeFromFee	External	✓	onlyOwner
	includeInFee	External	✓	onlyOwner

	excludeFromAMMs	External	✓	onlyOwner
	includeInAMMs	External	✓	onlyOwner
	changeLotteryPrize	External	✓	onlyOwner
		External	Payable	-
	_transfer	Internal	✓	
	swapAndLiquify	Private	✓	lockTheSwap
	random	Public		-
	swapTokensForEth	Private	✓	
	addLiquidity	Private	✓	
	_approve	Internal	✓	

Inheritance Graph



Flow Graph



Summary

Payout-coin 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 manipulate the fees. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a limit of max 30% 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