Cyberscope

Audit Report

Patrios

May 2024

SHA256 7d2617b58316cc99b2a0003f4a28d655a29c2b5bd63585253c5f8ab56067c5b0

Audited by © cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

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

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	EIS	Excessively Integer Size	Unresolved
●	IFC	Inconsistent Fee Calculations	Unresolved
●	IAM	Inefficient Approval Management	Unresolved
●	MEM	Misleading Error Messages	Unresolved
●	PMRM	Potential Mocked Router Manipulation	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	RAC	Redundant Amount Calculation	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L11	Unnecessary Boolean equality	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	Patriots_Coin
Testing Deploy	https://testnet.bscscan.com/address/0x09dc0853b8ee8df8a2be3767a809b72b172af818
Symbol	PTC
Decimals	18
Total Supply	17,760,000
Badge Eligibility	Must Fix Critical

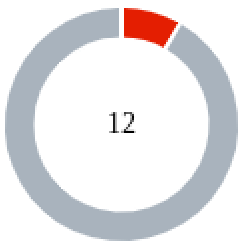
Audit Updates

Initial Audit	23 May 2024

Source Files

Filename	SHA256
contracts/PTC.sol	7d2617b58316cc99b2a0003f4a28d655a29c2b5bd63585253c5f8ab56067c5b0

Findings Breakdown



●	Critical	1
●	Medium	0
●	Minor / Informative	11

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

ST - Stops Transactions

Criticality	Critical
Location	contracts/PTC.sol
Status	Unresolved

Description

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

Recommendation

The team is advised to follow the recommendations outlined in the `PMRM` and `PTRP` findings and implement the necessary steps to mitigate the identified risk, ensuring that the contract does not operate as a honeypot.

EIS - Excessively Integer Size

Criticality	Minor / Informative
Location	contracts/PTC.sol#L68
Status	Unresolved

Description

The contract is using a bigger unsigned integer data type that the maximum size that is required. By using an unsigned integer data type larger than necessary, the smart contract consumes more storage space and requires additional computational resources for calculations and operations involving these variables. This can result in higher transaction costs, longer execution times, and potential scalability bottlenecks.

Specifically, given that the maximum value for the variables `totalTax` , `burnSplit` , and `marketingSplit` could be `300` , it would be more efficient to use a smaller unsigned integer type instead of uint256. For instace, using a `uint16` , which can hold values from 0 to 65535 will reduce the storage size and can save gas costs.

```
uint256 public totalTax;
uint256 public burnSplit;
uint256 public marketingSplit;
```

Recommendation

To address the inefficiency associated with using an oversized unsigned integer data type, it is recommended to accurately determine the required size based on the range of values the variable needs to represent.

IFC - Inconsistent Fee Calculations

Criticality	Minor / Informative
Location	PTC.sol#L134
Status	Unresolved

Description

The contract is setting specific amounts for various fee values (e.g., `burnSplit`, `marketingSplit`). However, the calculation of these fees is not consistently based on these specific amounts. Instead, the contract calculates percentage amounts based on other derived values, which can lead to inconsistencies and unexpected behavior in fee distribution.

```
if (burnSplit != 0)
    _burn(
        address(this),
        (balanceOf(address(this)) * burnSplit) / totalTax
    );
swapTokensForETH(balanceOf(address(this)));
if (address(this).balance > threshold) {
    if (marketingSplit != 0)
        marketingWallet.transfer(
            (address(this).balance * marketingSplit) /
                (totalTax - burnSplit)
        );
    if (!isBurnFinished) {
        burnPREME(
            (address(this).balance * 100) /
                (totalTax - burnSplit - marketingSplit)
        );
```

Recommendation

It is recommended to use a standardized percentage approach for fee distribution. This would ensure that all fee calculations are consistent and based directly on the specified fee amounts, improving clarity and predictability in the fee distribution process. By ensuring that all fee-related calculations are based on the specified percentages, the contract will maintain a clear and consistent logic for fee distribution.

IAM - Inefficient Approval Management

Criticality	Minor / Informative
Location	PTC.sol#L175
Status	Unresolved

Description

The contract is calling the `approve(address(router), ~uint256(0))` function, which approves the router to spend an unlimited amount of tokens, within every execution of the `swapTokensForETH` function. This practice is inefficient and could lead to increased gas costs due to repetitive approvals. Additionally, this may pose a security risk if the router's address changes or if the approval needs to be revoked for any reason.

```
   function swapTokensForETH(uint256 _amount) private {
        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = router.WETH();
        uint256 amountOutMin = getAmount(_amount, router,
path);
        // Approve router to spend tokens
        approve(address(router), ~uint256(0));
    ...
```

Recommendation

It is recommended to perform the `approve(address(router), ~uint256(0))` operation once during the initial setup phase or within a dedicated initialization function. This would avoid redundant approvals, thereby optimizing gas usage and simplifying the contract logic. By moving the approval out of the `swapTokensForETH` function and into an initialize function, the contract becomes more efficient and secure.

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	PTC.sol#L257
Status	Unresolved

Description

The contract is using misleading error messages. These error messages do not accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
require(inSwapAndLiquify)
```

Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

PMRM - Potential Mocked Router Manipulation

Criticality	Minor / Informative
Location	contracts/PTC.sol#L208
Status	Unresolved

Description

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

```
function setSwapRouter(IRouter _newAddress) public onlyOwner {
    require(router != _newAddress, "The router already has this
address");
    address pair = IFactory(_newAddress.factory()).getPair(
        address(this),
        _newAddress.WETH()
    );
    if (pair == address(0))
        pair = IFactory(_newAddress.factory()).createPair(
            address(this),
            _newAddress.WETH()
        );
    if (isPair[pair] != true && pair != address(0))
setPair(pair, true);
    _approve(address(this), address(_newAddress), ~uint256(0));
    swapPair = pair;
    router = _newAddress;
    emit SetSwapRouter(_newAddress);
}
```

Recommendation

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

Temporary Solutions:

These measurements do not decrease the severity of the finding

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

Permanent Solution:

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

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	contracts/PTC.sol#L142,154
Status	Unresolved

Description

The contract sends funds to a `marketingWallet` and `teamWallet` 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.

```
if (marketingSplit != 0)
    marketingWallet.transfer(
        (address(this).balance * marketingSplit) /
            (totalTax - burnSplit)
    );
...
}
if (address(this).balance != 0)
    teamWallet.transfer(address(this).balance);
```

Recommendation

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

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	contracts/PTC.sol#L120,138
Status	Unresolved

Description

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

```
   balanceOf(address(this)) > swapAtAmount &&
    isPair[to] &&
    !inSwapAndLiquify
) handleTax();
...
    function handleTax() private lockTheSwap {
        ...
        swapTokensForETH(balanceOf(address(this)));
        ...
```

Recommendation

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

RAC - Redundant Amount Calculation

Criticality	Minor / Informative
Location	PTC.sol#L173
Status	Unresolved

Description

The contract is redundantly calculating the `amountOutMin` variable by calling the `getAmount` function, which is unnecessary because the `swapExactTokensForETH` function will ensure that the contract receives the appropriate amount of ETH based on the token swap. This redundancy results in extra computational steps and increased gas consumption without providing additional benefits.

```
uint256 amountOutMin = getAmount(_amount, router, path);
// Approve router to spend tokens
approve(address(router), ~uint256(0));
try
    router.swapExactTokensForETH(
        balanceOf(address(this)),
        amountOutMin,
        path,
        address(this),
        block.timestamp
    )
```

Recommendation

It is recommended to remove the calculation of the `amountOutMin` variable to streamline the code and reduce gas costs. This can be achieved by directly using the router's functionality to handle the token swap without pre-calculating the minimum output amount. By setting `amountOutMin` to 0, the contract will proceed with the token swap regardless of the exact ETH amount received, thereby simplifying the logic and reducing unnecessary gas usage.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/PTC.sol#L9,46,79,82,160,161,162,169,191,208,227,234,244,255,256,257,274,275,290,304,318
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);

...
```

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.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	contracts/PTC.sol#L219,295,309,319
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
isPair[pair] != true && pair != address(0)
ExcludedFromTax[marketingWallet] == true)
ExcludedFromTax[teamWallet] == true)
ExcludedFromTax[owner()] == true)
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/PTC.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 ^0.8.25;
```

Recommendation

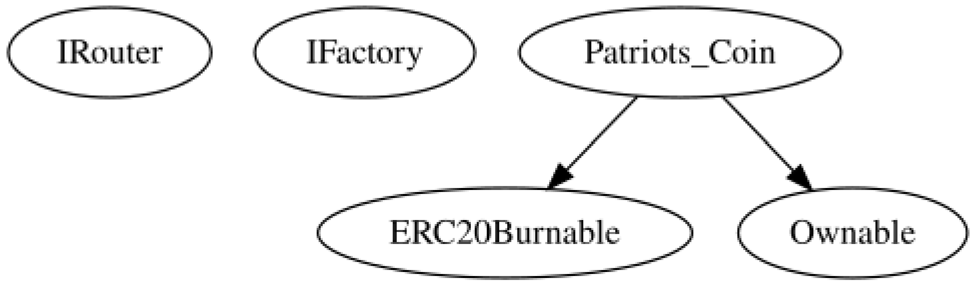
The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest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

Functions Analysis

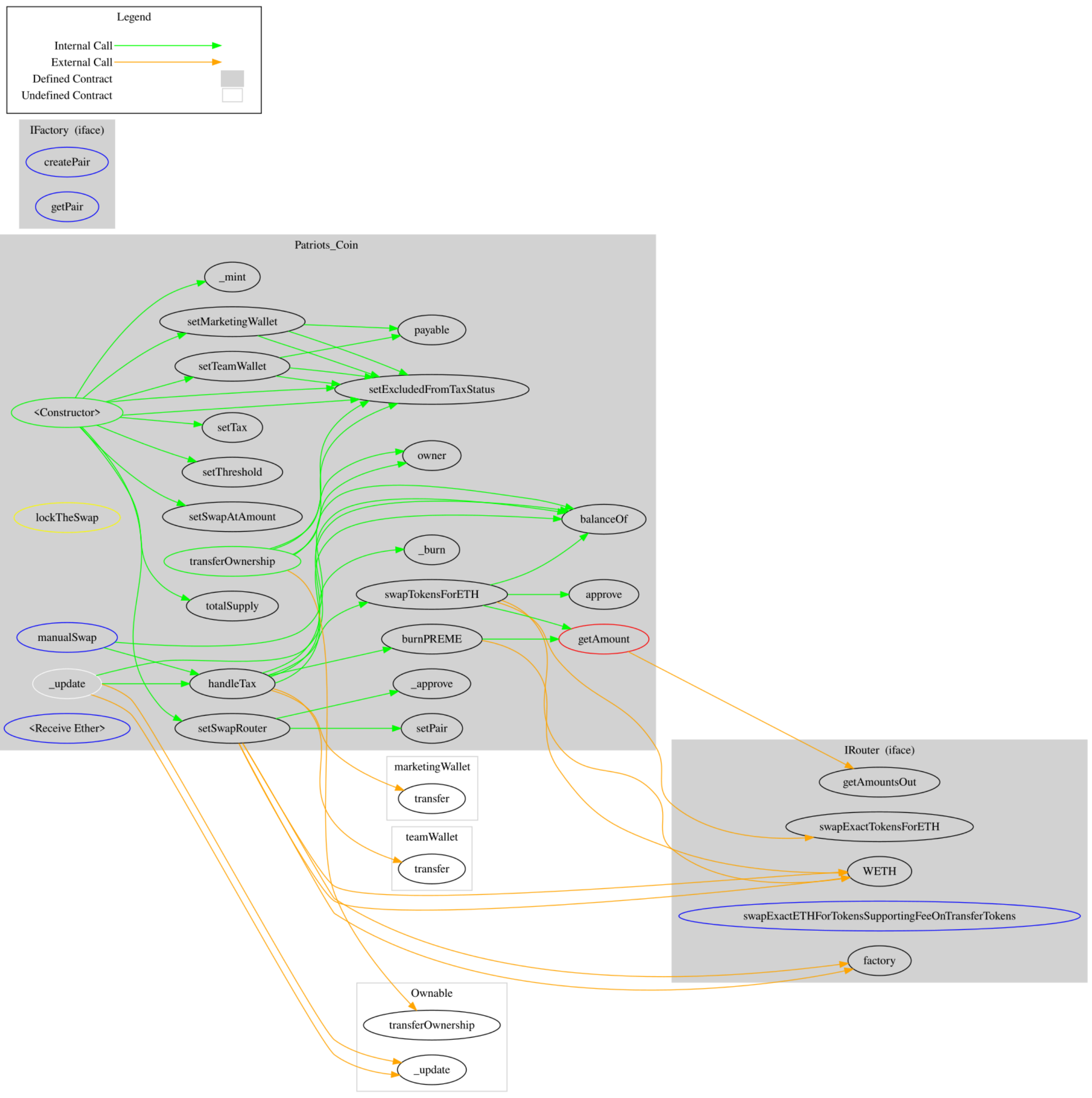
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Patriots_Coin	Implementation	ERC20Burnable, Ownable		
		Public	✓	ERC20 Ownable
	_update	Internal	✓	
	handleTax	Private	✓	lockTheSwap
	getAmount	Private		
	swapTokensForETH	Private	✓	
	burnPREME	Private	✓	
	setSwapRouter	Public	✓	onlyOwner
	setPair	Public	✓	onlyOwner
	setSwapAtAmount	Public	✓	onlyOwner
	setThreshold	Public	✓	onlyOwner
	setTax	Public	✓	onlyOwner
	setExcludedFromTaxStatus	Public	✓	onlyOwner
	setMarketingWallet	Public	✓	onlyOwner
	setTeamWallet	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner

	manualSwap	External	✓	onlyOwner
		External	Payable	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IFactory (iface)
createPair
getPair
Patriots_Coin
_mint
setMarketingWallet
payable
setTeamWallet
setExcludedFromTaxStatus
<Constructor>
setTax
setThreshold
owner
lockTheSwap
setSwapAtAmount
balanceOf
transferOwnership
_burn
totalSupply
swapTokensForETH
approve
manualSwap
burnPREME
getAmount
_update
handleTax
_approve
<Receive Ether>
setSwapRouter
setPair
marketingWallet
transfer
teamWallet
transfer
IRouter (iface)
getAmountsOut
swapExactTokensForETH
WETH
swapExactETHForTokensSupportingFeeOnTransferTokens
factory
Ownable
transferOwnership
_update

Summary

Patrios contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner like stop transactions. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a maximum 3% fee limit for buy and sell transactions. The contract implements a buy and burn functionality of the PREME token, which is set to automatically end with the next distribution after November 4th 2024, 11:59:59 PM EST.

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