Cyberscope

Audit Report

PVPFUN.tech

April 2025

Network	ETH
Address	0xC3B8Bc9Baef8D217aB101d9Cc95fb7547e2b64fF
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
●	UAR	Unexcluded Address Restrictions	Unresolved
●	HV	Hardcoded Values	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	MC	Missing Check	Unresolved
●	MEM	Missing Error Messages	Unresolved
●	NWES	Nonconformity with ERC-20 Standard	Unresolved
●	PGA	Potential Griefing Attack	Unresolved
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	RRA	Redundant Repeated Approvals	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
● Critical	Highly Likely / High Impact
● Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
● Minor / Informative	Unlikely / Low to no Impact

Review

Contract Name	PVPFUN
Compiler Version	v0.8.23+commit.f704f362
Optimization	200 runs
Explorer	https://etherscan.io/address/0xc3b8bc9baef8d217ab101d9cc95fb7547e2b64ff
Address	0xc3b8bc9baef8d217ab101d9cc95fb7547e2b64ff
Network	ETH
Symbol	PVPFUN
Decimals	18
Total Supply	1.000.000.000

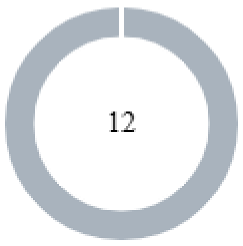
Audit Updates

Initial Audit	21 Apr 2025

Source Files

Filename	SHA256
PVPFUN.sol	210c9989c7a6d9ae6885ead67e05eaa4719407af15685ad416c94e31e4012286

Findings Breakdown



●	Critical	0
●	Medium	0
●	Minor / Informative	12

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	0	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	12	0	0	0

UAR - Unexcluded Address Restrictions

Criticality	Minor / Informative
Location	PVPFUN.sol#L202,217
Status	Unresolved

Description

The contract incorporates operational restrictions on transactions, which can hinder seamless interaction with decentralized applications (dApps) such as launchpads, presales, lockers, or staking platforms. In scenarios where an external contract, such as a launchpad factory, needs to integrate with the contract, it should be exempt from the limitations to ensure uninterrupted service and functionality. Failure to provide such exemptions can block the successful process and operation of services reliant on this contract.

```
function _transfer(address from, address to, uint256 amount)
private {
        //...
        if (!_isExcludedFromFee[from] &&
!_isExcludedFromFee[to]) {
            //...
            if(from == uniswapV2Pair && to != address(this)){
                taxAmount = amount * _BuyTax / 100;
            }
            if(to == uniswapV2Pair && from != address(this)){
                taxAmount = amount * _SellTax / 100;
            }
            uint256 contractTokenBalance =
balanceOf(address(this));
            if (!inSwap && to == uniswapV2Pair && swapEnabled
&& contractTokenBalance>_taxSwapThreshold) {
                if (block.number > lastContractSellBlock) {
                    contractSellCount = 0;
                }
                require(contractSellCount < 2);
                //...
                contractSellCount++;
                lastContractSellBlock = block.number;
            }
        }
        if(taxAmount>0){
          _balances[address(this)] += taxAmount;
          emit Transfer(from, address(this),taxAmount);
        }
        _balances[from] = _balances[from] - amount;
        _balances[to] = _balances[to] + (amount - taxAmount);
        emit Transfer(from, to, amount - taxAmount);
    }
```

Recommendation

It is advisable to modify the contract by incorporating functionality that enables the exclusion of designated addresses from transactional restrictions. This enhancement will allow specific addresses, such as those associated with decentralized applications (dApps) and service platforms, to operate without being hindered by the standard constraints imposed on other users. Implementing this feature will ensure smoother integration and functionality with external systems, thereby expanding the contract's versatility and effectiveness in diverse operational environments.

HV - Hardcoded Values

Criticality	Minor / Informative
Location	PVPFUN.sol#L205
Status	Unresolved

Description

The contract contains instances where numeric values are directly hardcoded into the code logic rather than being assigned to constant variables with descriptive names. Hardcoding such values can lead to several issues, including reduced code readability, increased risk of errors during updates or maintenance, and difficulty in consistently managing values throughout the contract. Hardcoded values can obscure the intent behind the numbers, making it challenging for developers to modify or for users to understand the contract effectively.

```
require(contractSellCount < 2);
```

Recommendation

It is recommended to replace hardcoded numeric values with variables that have meaningful names. This practice improves code readability and maintainability by clearly indicating the purpose of each value, reducing the likelihood of errors during future modifications. Additionally, consider using constant variables which provide a reliable way to centralize and manage values, improving gas optimization throughout the contract.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	PVPFUN.sol#L118,119
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.

```
_devWallet
_marketingWallet
```

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.

MC - Missing Check

Criticality	Minor / Informative
Location	PVPFUN.sol#L221
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

In `_transfer` function a check and error message is missing to ensure that the `balance` of the sender is greater than or equal to the `amount` to be sent.

```
_balances[from] = _balances[from] - amount;
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

MEM - Missing Error Messages

Criticality	Minor / Informative
Location	PVPFUN.sol#L205
Status	Unresolved

Description

The contract is missing error messages. Specifically, there are no error messages to 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(contractSellCount < 2)
```

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.

NWES - Nonconformity with ERC-20 Standard

Criticality	Minor / Informative
Location	PVPFUN.sol#L180
Status	Unresolved

Description

The contract does not fully conform to the ERC20 Standard. Specifically, according to the standard, transfers of 0 values must be treated as normal transfers and fire the Transfer event. However, the contract implements a conditional check that prohibits transfers of 0 values. This discrepancy between the contract's implementation and the ERC20 standard may lead to inconsistencies and incompatibilities with other contracts.

```
function _transfer(address from, address to, uint256 amount)
private {
        //...
        require(amount > 0, "Transfer amount must be greater
than zero");
        //...
}
```

Recommendation

The incorrect implementation of the ERC20 standard could potentially lead to problems when interacting with the contract, as other contracts or applications that expect the ERC20 interface may not behave as expected. The team is advised to review and revise the implementation of the transfer mechanism to ensure full compliance with the ERC20 standard. https://eips.ethereum.org/EIPS/eip-20.

PGA - Potential Griefing Attack

Criticality	Minor / Informative
Location	PVPFUN.sol#L205
Status	Unresolved

Description

The contract includes functionality designed to enforce specific conditions on transactions. However, this design is vulnerable to griefing attacks, where malicious actors can exploit the contract's logic to interfere with legitimate user operations.

`_transfer` function limits the amount of times the contract can swap in the same block by checking if the `contractSellCount` is less than 2. A third party could strategically interact with the contract's state to disrupt normal user operations, resulting in failed transactions or unintended behavior.

Such griefing attacks could undermine the contract's usability and obstruct legitimate user operations.

```
if (!inSwap && to == uniswapV2Pair && swapEnabled &&
contractTokenBalance>_taxSwapThreshold) {
    if (block.number > lastContractSellBlock) {
        contractSellCount = 0;
    }
    require(contractSellCount < 2);
    //...
    contractSellCount++;
    lastContractSellBlock = block.number;
}
```

Recommendation

The team is advised to review the transfer mechanism to ensure that all legitimate operations are processed as intended. This will help maintain the integrity of user activities and strengthen trust in the system.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	PVPFUN.sol#L233
Status	Unresolved

Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

```
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTok
ens(tokenAmount, 0, path, address(this), block.timestamp);
```

Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

RRA - Redundant Repeated Approvals

Criticality	Minor / Informative
Location	PVPFUN.sol#L232
Status	Unresolved

Description

The contract is designed to `approve` token transfers during the contract's operation by calling the _approve function before specific operations. This approach results in additional gas costs since the approval process is repeated for every operation execution, leading to inefficiencies and increased transaction expenses.

```
function swapTokensForEth(uint256 tokenAmount) private
lockTheSwap {
    //...
    _approve(address(this), address(uniswapV2Router),
tokenAmount);

uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTok
ens(
        tokenAmount,
        0,
        path,
        address(this),
        block.timestamp
    );
}
```

Recommendation

Since the approved address is a trusted third-party source, it is recommended to optimize the contract by approving the maximum amount of tokens once in the initial set of the variable, rather than before each operation. This change will reduce the overall gas consumption and improve the efficiency of the contract.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	PVPFUN.sol#L98,99
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 _taxSwapThreshold= _tTotal * 5 / 10000;
uint256 public _maxTaxSwap= _tTotal * 1 / 100;
```

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	PVPFUN.sol#L90,91,92,93,94,95,96,97,98,99,262
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.

```
string private constant _name =     unicode"pvpfun.tech";
string private constant _symbol =   unicode"PVPFUN";
uint8 private constant _decimals = 18;
uint256 private constant _tTotal = 1000000000 * 10**_decimals;
uint256 public _BuyTax=            15;
uint256 public _SellTax=           35;
uint256 public _maxTxAmount =       _tTotal * 20 / 1000;
uint256 public _maxWalletSize =     _tTotal * 30 / 1000;
uint256 public _taxSwapThreshold=   _tTotal * 5 / 10000;
uint256 public _maxTaxSwap=         _tTotal * 1 / 100;
uint256 BuyTax
uint256 SellTax
```

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.solidityland.org/en/stable/style-guide.html#naming-conventions.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	PVPFUN.sol#L87,118
Status	Unresolved

Description

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

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

```
_devWallet
```

Recommendation

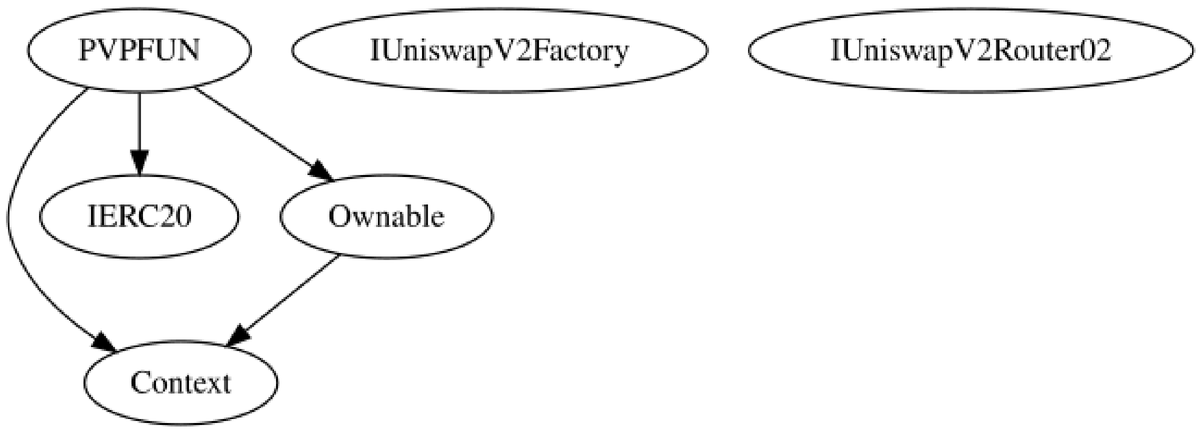
To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

Functions Analysis

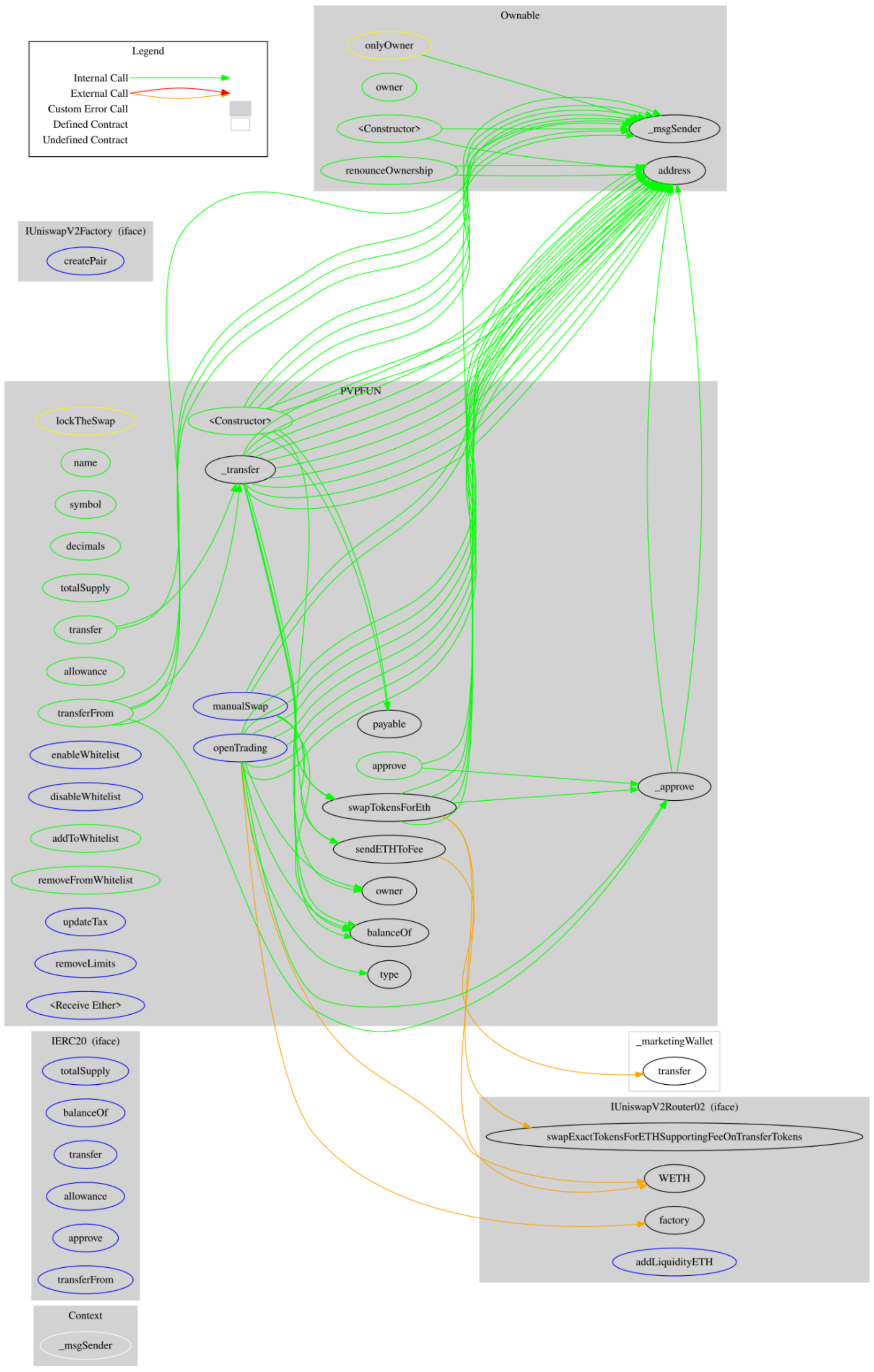
Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
PVPFUN	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	_approve	Private	✓	
	_transfer	Private	✓	
	swapTokensForEth	Private	✓	lockTheSwap
	enableWhitelist	External	✓	onlyOwner
	disableWhitelist	External	✓	onlyOwner
	addToWhitelist	Public	✓	onlyOwner
	removeFromWhitelist	Public	✓	onlyOwner
	updateTax	External	✓	onlyOwner
	removeLimits	External	✓	onlyOwner

	openTrading	External	✓	onlyOwner
	manualSwap	External	✓	onlyOwner
	sendETHToFee	Private	✓	
		External	Payable	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
Ownable
onlyOwner
owner
<Constructor>
renounceOwnership
_msgSender
address
IUniswapV2Factory (iface)
createPair
PVPFUN
lockTheSwap
name
symbol
decimals
totalSupply
transfer
allowance
transferFrom
enableWhitelist
disableWhitelist
addToWhitelist
removeFromWhitelist
updateTax
removeLimits
<Receive Ether>
<Constructor>
_transfer
manualSwap
openTrading
payable
approve
swapTokensForEth
sendETHToFee
owner
balanceOf
type
_approve
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Context
_msgSender
_marketingWallet
transfer
IUniswapV2Router02 (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
WETH
factory
addLiquidityETH

Summary

PVPFUN.tech contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. PVPFUN.tech is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract has 5% fees.

The contract has renounced the ownership so it no longer has an assigned owner and consequently, the owner's privileges and authority are revoked. As a result, the owner is unable to execute any methods that are designated exclusively for owner access. By relinquishing ownership, the contract eliminates the potential risks associated with centralized authority, reducing the possibility of the owner misusing their privileges or becoming a single point of failure. It is important to note that renouncing ownership is an irreversible action, and once executed, it cannot be undone.

The ownership has been renounced on this transaction:

https://etherscan.io/tx/0x6dc0887b2c2d2dbabb0518f96a19dec259b2894b7da9490f55709e33bea1c99d

Disclaimer

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

cyberscope.io