Cyberscope

Audit Report

CHAMP

July 2025

SHA256 :

c99887491033bdc80a2fd864f47bfb203149ca4b7d8e3dd551e01c7d7dc55282

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
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	MVN	Misleading Variables Naming	Unresolved
●	PMRM	Potential Mocked Router Manipulation	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	RCS	Redundant Conditional Statements	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	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

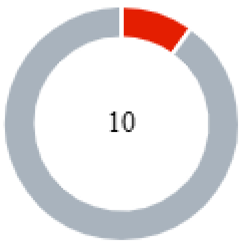
Audit Updates

Initial Audit	09 Mar 2025 https://github.com/cyberscope-io/audits/blob/main/4-ccg/v1/audit.pdf
Corrected Phase 2	21 Mar 2025 https://github.com/cyberscope-io/audits/blob/main/4-ccg/v2/audit.pdf
Corrected Phase 3	22 Apr 2025 https://github.com/cyberscope-io/audits/blob/main/4-ccg/v3/audit.pdf
Corrected Phase 4	15 May 2025 https://github.com/cyberscope-io/audits/blob/main/4-ccg/v4/audit.pdf
Corrected Phase 5	19 Jun 2025 https://github.com/cyberscope-io/audits/blob/main/4-ccg/v5/audit.pdf
Corrected Phase 6	09 Jul 2025 https://github.com/cyberscope-io/audits/blob/main/4-ccg/v6/audit.pdf
Corrected Phase 7	22 Jul 2025
Test Deploys	https://sepolia.etherscan.io/address/0xc140924fcbef24498d3f8a4b35538f4a38744427

Source Files

Filename	SHA256
CChamps-July-16th.sol	c99887491033bdc80a2fd864f47bfb203149ca4b7d8e3dd551e01c7d7dc55282

Findings Breakdown



●	Critical	1
●	Medium	0
●	Minor / Informative	9

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	1	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	9	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	CChamps_July-6th.sol#L167
Status	Unresolved

Description

The contract can stop the sales for all users. The sales can be stoped as described in the finding `PTRP`. As a result, the contract may operate as a honeypot.

```
_ethSendToMarketing(wethOutFromSwap - wethUsedInLiquidity);
```

Recommendation

The contract should embody the necessary checks to prevent transactions from stoping. This can be resolved as described in the finding `PTRP`.

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.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	CChamps_July-6th.sol#L163,227
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.

```
uint256 wethUsedInLiquidity = _addToLiquidity(
    ethForLiquidity,
    liquidityHalf
);
...
function _addToLiquidity(
uint256 amountInEth,
uint256 amountInToken
) private nonReentrant returns (uint256) {
uint256 initialBalance = address(this).balance;
uniswapRouter.addLiquidityETH{value: amountInEth}(
    address(this),
    amountInToken,
    0, // slippage is unavoidable
    0, // slippage is unavoidable
    owner(),
    block.timestamp
);
return initialBalance - address(this).balance;
}
```

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.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	CChamps_July-6th.sol#L104
Status	Unresolved

Description

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

```
function changeSwapThreshold(uint256 _tokens) external onlyOwner {
require(_tokens > 0, "Minimum token value must be higher than 0");
swapThresholdAmount = _tokens;
emit SwapThresholdUpdated(_tokens);
}
```

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.

MVN - Misleading Variables Naming

Criticality	Minor / Informative
Location	CChamps_July-6th.sol#L181
Status	Unresolved

Description

Variables can have misleading names if their names do not accurately reflect the value they contain or the purpose they serve. The contract uses some variable names that are too generic or do not clearly convey the information stored in the variable. Misleading variable names can lead to confusion, making the code more difficult to read and understand. In this case, the `buyTax` is invoked for all taxed transactions, even if they are not buys.

```
uint256 transactionTax = (to == liquidityPool) ? sellTax : buyTax;
```

Recommendation

It's always a good practice for the contract to contain variable names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

PMRM - Potential Mocked Router Manipulation

Criticality	Minor / Informative
Location	CChamps_July-6th.sol#L81
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.

```
constructor(
address _uniswapRouter,
address _marketingWallet
) ERC20("CryptoChamps", "CCG") Ownable(_msgSender())
```

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	CChamps_July-6th.sol#L167
Status	Unresolved

Description

The contract sends funds to a `marketingWallet` as part of the transfer flow. This address can either be a wallet address or a contract. If the address belongs to a contract then it may revert from incoming payment. As a result, the error will propagate to the token's contract and revert the transfer. In addition, the receiver may execute custom logic that consumes the gas of the transaction, effectively preventing it from finalizing.

```
function _update(
address from,
address to,
uint256 amount
) internal override {
...
_ethSendToMarketing(wethOutFromSwap - wethUsedInLiquidity);
...
}
```

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.

RCS - Redundant Conditional Statements

Criticality	Minor / Informative
Location	CChamps_July-6th.sol#L146,151
Status	Unresolved

Description

The contract contains redundant conditional statements that can be simplified to improve code efficiency and performance. Conditional statements that are always true are unnecessary and lead to larger code size, increased memory usage, and slower execution times. Specifically, the `totalAllocation` is always greater than zero and the `liquidityTax` will never surpass the `contractTokenBalance` .

```
require(totalAllocation > 0, "Allocations must be > 0");
...
require(
liquidityTax <= contractTokenBalance,
"Invalid allocation: exceeds balance"
);
```

Recommendation

It is recommended to refactor conditional statements for redundancies. This practice minimizes the number of operations required, reduces the code footprint, and optimizes memory and gas usage. Simplifying such statements makes the code more readable and improves its overall performance.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	CChamps-July-16th.sol#L104,110,267,279,293,294
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.

```
uint256 _tokens
bool _applyTax
address _marketingWallet
uint256 _buyTax, uint256 _sellTax
uint256 _liquidityAllocation
uint256 _marketingAllocation
```

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/stable/style-guide.html#naming-conventions.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	CChamps-July-16th.sol#L155,161
Status	Unresolved

Description

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

```
uint256 liquidityHalf = liquidityTax / 2;

uint256 ethForLiquidity = (wethOutFromSwap * liquidityHalf) /
tokensToSwap;
```

Recommendation

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

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	CChamps-July-16th.sol#L92
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.

```
marketingWallet = _marketingWallet
```

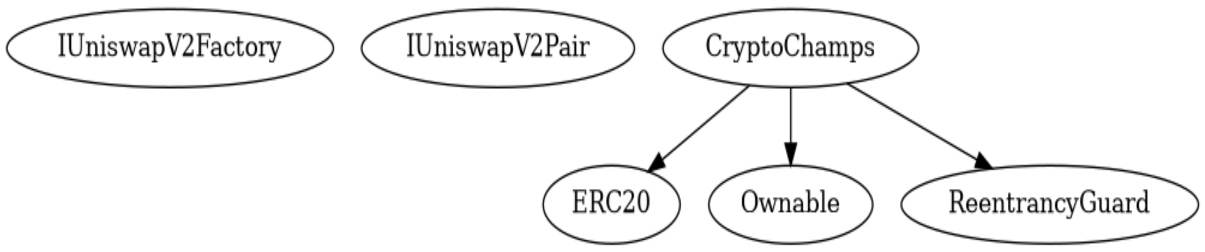
Recommendation

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.

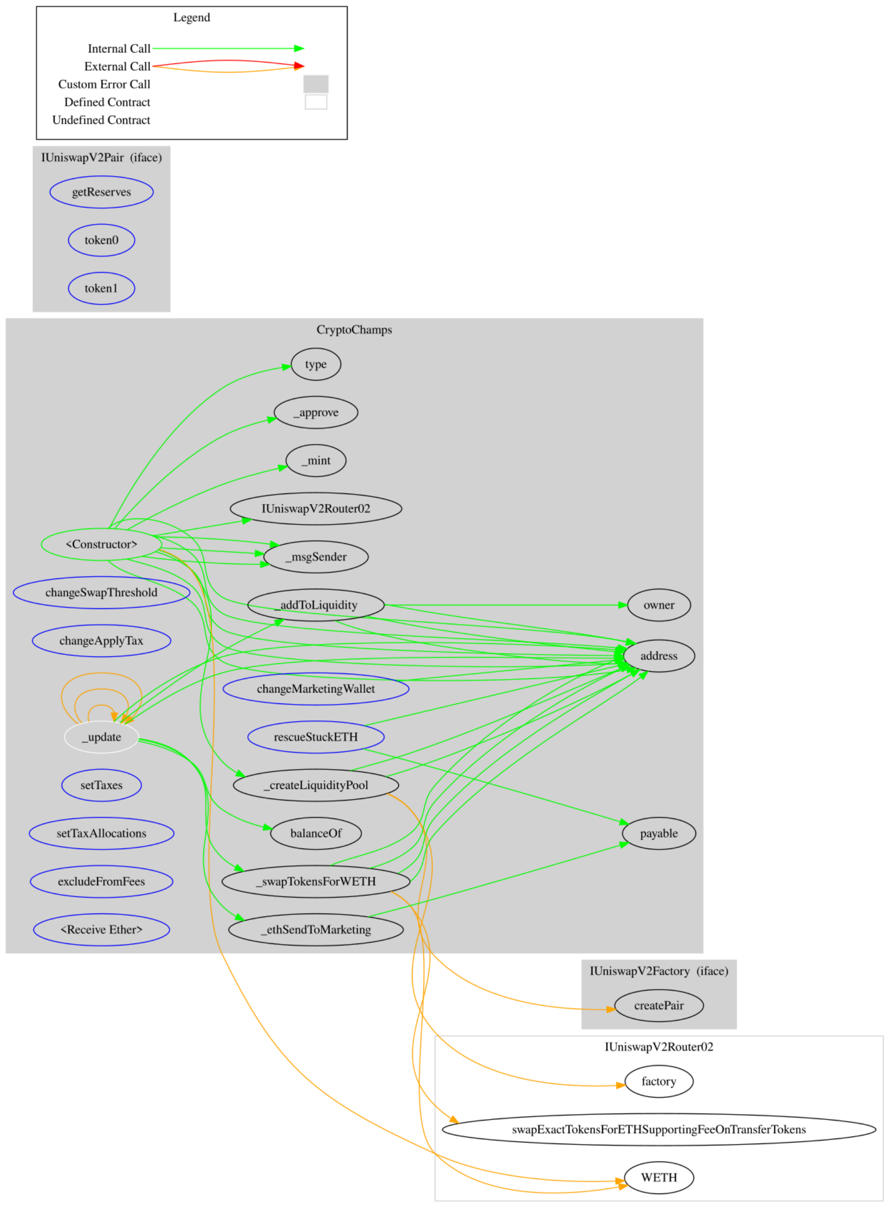
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
CryptoChamps	Implementation	ERC20, Ownable, ReentrancyGuard		
		Public	✓	ERC20 Ownable
	changeSwapThreshold	External	✓	onlyOwner
	changeApplyTax	External	✓	onlyOwner
	_createLiquidityPool	Internal	✓	
	_update	Internal	✓	
	_swapTokensForWETH	Private	✓	nonReentrant
	_addToLiquidity	Private	✓	nonReentrant
	_ethSendToMarketing	Internal	✓	
	changeMarketingWallet	External	✓	onlyOwner
	setTaxes	External	✓	onlyOwner
	setTaxAllocations	External	✓	onlyOwner
	excludeFromFees	External	✓	onlyOwner
	rescueStuckETH	External	✓	onlyOwner
		External	Payable	-

Inheritance Graphs



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
IUniswapV2Pair (iface)
getReserves
token0
token1
CryptoChamps
type
_approve
_mint
IUniswapV2Router02
<Constructor>
_msgSender
changeSwapThreshold
_addToLiquidity
owner
changeApplyTax
address
changeMarketingWallet
_update
rescueStuckETH
setTaxes
_createLiquidityPool
setTaxAllocations
balanceOf
payable
excludeFromFees
_swapTokensForWETH
<Receive Ether>
_ethSendToMarketing
IUniswapV2Factory (iface)
createPair
IUniswapV2Router02
factory
swapExactTokensForETHSupportingFeeOnTransferTokens
WETH

Summary

CryptoChamps 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 limit of max 10% fees.

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