Cyberscope

Audit Report

CHAMP

May 2025

SHA256 :

e6bf95556b4c6c080ce17c31ae1c6d0e348d42a1239e0d2b08fc3ff9f74e26ba

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
Medium	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
Minor / Informative	CRC	Cyclic Reward Claim	Unresolved
Minor / Informative	FLV	Flash Loan Vulnerability	Unresolved
Minor / Informative	MVN	Misleading Variables Naming	Unresolved
Minor / Informative	PMRM	Potential Mocked Router Manipulation	Unresolved
Minor / Informative	PUR	Potentially Unclaimable Rewards	Unresolved
Minor / Informative	PVC	Price Volatility Concern	Unresolved
Minor / Informative	RSV	Redundant State Variables	Unresolved
Minor / Informative	USCR	Unclaimable Smart Contract Rewards	Unresolved
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor / Informative	L07	Missing Events Arithmetic	Unresolved
Minor / Informative	L13	Divide before Multiply Operation	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

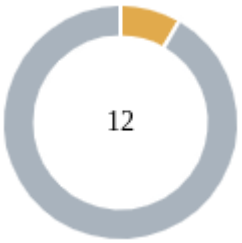
Audit Updates

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
Test Deploys	https://sepolia.etherscan.io/address/0x2CC63A049B262D91437dcCad73acC12DF7C880C6

Source Files

Filename	SHA256
CryptoChamps.sol	e6bf95556b4c6c080ce17c31ae1c6d0e348d42a1239e0d2b08fc3ff9f74e26ba

Findings Breakdown



Critical	0
Medium	1
Minor / Informative	11

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

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Medium
Location	CChampsV2.sol#L367,447
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.

```
function _swapTokensForWETH(
uint256 tokenAmount
) private nonReentrant returns (uint256) {
if (tokenAmount == 0) return 0;
(uint112 r0, uint112 r1, ) = IUniswapV2Pair(liquidityPool)
.getReserves();
require(r0 > 1e18 && r1 > 1e18, "No liquidity");
...}
```

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.

CRC - Cyclic Reward Claim

Criticality	Minor / Informative
Location	CChampsV2.sol#L565
Status	Unresolved

Description

The contract implements a mechanism to claim rewards. Rewards are calculated proportionally to the user's balance. Users are able to claim within 30 second intervals. During subsequent claims users receive rewards excluding the already claimed amounts. However users are able to transfer their tokens to new wallets and reset that record. This would enable users to withdraw larger rewards that expected.

```
function claimReflections() public nonReentrant onlyEOA {
...
owed = ((balanceOf(msg.sender) *
cumulativeReflectionsPerToken) / 1e18) -
reflectionDebt[msg.sender];
...
```

Recommendation

The team is advised to accommodate for such cases where users can effectively bypass all restrictions. The team is advised to employ a staking mechanism if rewards are to be distributed proportionally to a user's balance. This would ensure that all amounts reflect the actual user balance.

FLV - Flash Loan Vulnerability

Criticality	Minor / Informative
Location	CChampsV2.sol#L565
Status	Unresolved

Description

The `claimReflections` function is susceptible to flash-loan attacks. Specifically, the function calculates the `reflectionShare` based on the user balance. A flash loan allows a user to borrow a large amount of tokens from a liquidity pool, perform operations, and return them within the same transaction. In this scenario, a user could borrow a large amount of tokens, to increase their balance. The use of `onlyEOA` does not suffice to prevent the attack. Specifically, the caller may be a smart contract that executes the attack within the constructor. At that stage the code length of the contract is zero. Alternatively, a user may deploy a smart contract that requests the smart contract. Then a batched transaction can be submitted that initiates the flash loan to the smart contract, with the funds being forwarded to the EOA. The latter can then call the `claimReflections` and subsequently return the borrowed assets to the contract which closes the flash loan. Within this batch the EOA exploits the vulnerability without the need to deploy code that interacts with the contract.

```
function calculateETHWillBeClaimable(
address holder
) public view returns (uint256) {
uint256 holderBalance = balanceOf(holder);
if (holderBalance < minimumHoldingForReflection) {
return 0;
}
uint256 taxBalance = balanceOf(address(this));
uint256 withoutLiquidity = (taxBalance * 100) /
reflectionAllocation;
(uint256 willReceived, ) = _wETHAmountAndPath(withoutLiquidity);
uint256 totalSupplyExcludingBurned = totalSupply() -
balanceOf(address(0));
uint256 reflectionShare = (holderBalance *
(totalReflectionsAccumulated + willReceived)) /
totalSupplyExcludingBurned;
uint256 alreadyClaimed = totalEthReflections[holder];
return reflectionShare - alreadyClaimed;
}
```

Recommendation

The team is advised to revise the implementation of the reward distribution mechanism to ensure secure operations. Specifically it is advised to consider implementing a staking mechanism if rewards are to be claimed proportionally to a user's balance. This will prevent flash loan attacks, as funds will remain locked within the contract for at least a block.

MVN - Misleading Variables Naming

Criticality	Minor / Informative
Location	CChampsV2.sol#L331
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	CChampsV2.sol#L143
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 _admin
) ERC20("Champ", "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.

PUR - Potentially Unclaimable Rewards

Criticality	Minor / Informative
Location	CChampsV2.sol#L565
Status	Unresolved

Description

The contract maintains an internal `reflectionDebt` variable for the users. This variable is used to track the amount of rewards eligible users may claim. In the current impementation there is a limitation of the necessary balance of tokens needed for a user to claim. Specifically, consider the following case. In this example, a user transfers part of their balance which triggers an update for the `reflectionDebt`. A subsequent call to the `claimReflections` method will fail due to an underflow. The contract therefore assumes that a user who transfers tokens has already claimed the total of their rewards which may not be true.

```
balanceOf(user) = 10;
\\The user transfers 7 tokens to userB
reflectionDebt[user] ~= 10;
```

```
\\The user calls claimReflections
owed = ((3 * 1e18) / 1e18) - 10 = -6;
```

Recommendation

The team is advised to monitor code behavior under all instances of the execution flow. A potential solution would include claiming all eligible rewards at the time of the transaction. This would ensure consistency of operation and enhance user trust.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	CChampsV2.sol#L215
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `swapTokensAtAmount` 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");swapTokensAtAmount = _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.

RSV - Redundant State Variables

Criticality	Minor / Informative
Location	CChampsV2.sol#L249
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 state variable `lastTransferBlock` is set and never utilized .

```
lastTransferBlock[from] = block.number;
lastTransferBlock[to] = block.number;
```

Recommendation

The team is advised to take into consideration these segments 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 execution.

USCR - Unclaimable Smart Contract Rewards

Criticality	Minor / Informative
Location	CChampsV2.sol#L483
Status	Unresolved

Description

The contract implements a reward distribution mechanism with rewards being allocated to externally owned accounts (EOAs) in proportion to their token balances. However, if a significant share of the total token supply is held by smart contracts,such as liquidity pool contracts, a substantial portion of the rewards may remain unclaimed. This is due to the use of the total supply in calculating `cumulativeReflectionsPerToken`, despite the fact that smart contracts are not permitted to claim rewards. As a result, a portion of the funds may not be distributed to users as intended.

```
function _distributeReflections(uint256 intendedAmount) private {
if (totalSupply() == 0) return;
reflectionEthBalance += intendedAmount;
cumulativeReflectionsPerToken +=
(intendedAmount * 1e18) /
totalSupply();
emit ReflectionsDistributed(intendedAmount);
}
```

Recommendation

The team should consider refactoring the implementation logic to consistently account for the balances held by known contracts, such as liquidity pool contracts. This adjustment would help ensure a more accurate and equitable distribution of rewards to the active users of the system.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	CChampsV2.sol#L5,89,187,201,215,619,637
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.

```
import "@uniswap/v2-periphery/contracts/interfac

ngeMinimumHoldingForReflection(

        uint256 _tokens

address _newAdmin
uint256 _tokens
uint256 _sellTax
uint256 _buyTax
uint256 _liquidityAllocation
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	CChampsV2.sol#L219
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.

```
swapTokensAtAmount = _tokens
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	CChampsV2.sol#L279,297,381,383,533,543
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

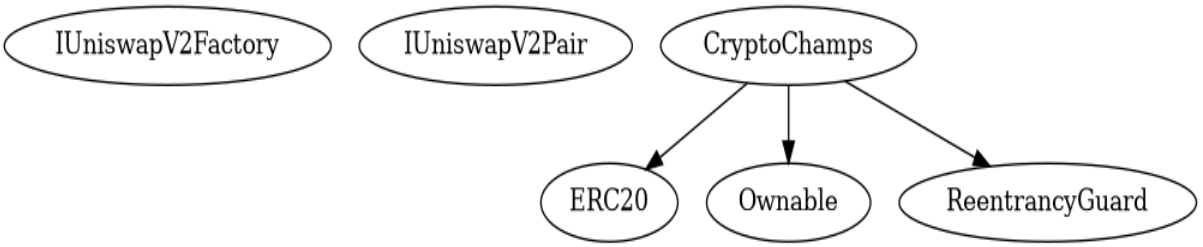
To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IUniswapV2Factory	Interface			
	createPair	External	✓	-
IUniswapV2Pair	Interface			
	getReserves	External	✓	-
CryptoChamps	Implementation	ERC20, Ownable, ReentrancyGuard		
		Public	✓	ERC20 Ownable
	changeAdmin	External	✓	onlyOwner
	changeMinimumHoldingForReflection	External	✓	onlyOwner
	changeSwapThreshold	External	✓	onlyOwner
	_createLiquidityPool	Internal	✓	
	_update	Internal	✓	
	_swapTokensForWETH	Private	✓	nonReentrant
	_wETHAmountAndPath	Private		
	_addToLiquidity	Private	✓	nonReentrant
	_distributeReflections	Private	✓	
	calculateETHWillBeClaimable	Public		-

	claimReflections	Public	✓	nonReentrant onlyEOA
	setTaxes	External	✓	onlyOwner
	setTaxAllocations	External	✓	onlyOwner
	excludeFromFees	External	✓	onlyOwner
		External	Payable	-

Inheritance Graphs



Summary

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

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

cyberscope.io