Cyberscope

Audit Report

CHAMP

September 2025

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
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	CRC	Cyclic Reward Claim	Unresolved
●	FLV	Flash Loan Vulnerability	Unresolved
●	MVN	Misleading Variables Naming	Unresolved
●	PMRM	Potential Mocked Router Manipulation	Unresolved
●	PSU	Potential Subtraction Underflow	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	RCS	Redundant Conditional Statements	Unresolved
●	USCR	Unclaimable Smart Contract Rewards	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	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

Contract Name	CHAMP
Symbol	CCG
Decimals	18
Total Supply	2,500,000,000
Badge Eligibility	Yes

Audit Updates
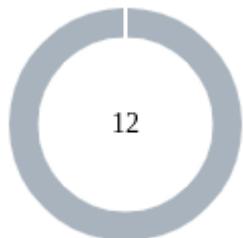
Initial Audit	09 Mar 2025
	https://github.com/cyberscope-io/audits/blob/main/4-ccg/v1/audit.pdf
Corrected Phase 2	21 Mar 2025
	https://github.com/cyberscope-io/audits/blob/main/4-ccg/v2/audit.pdf
Corrected Phase 3	22 Apr 2025
	https://github.com/cyberscope-io/audits/blob/main/4-ccg/v3/audit.pdf
Corrected 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
	https://github.com/cyberscope-io/audits/blob/main/4-ccg/v7/audit.pdf
Corrected Phase 8	26 Jul 2025
	https://github.com/cyberscope-io/audits/blob/main/4-ccg/v8/audit.pdf
Corrected Phase 9	17 Sep 2025

Source Files

Filename	SHA256
Official_Champ_Reflection_Token-9-13-25.sol	e2c1b7940ff5cfe3c50e0b9c337d2a3d7a1a25a17546f24d5c2290e53cd5f8db

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

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	Official_Champ_Reflection_Token-9-13-25.sol#L251
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.

```Shell
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.

CRC - Cyclic Reward Claim

Criticality	Minor / Informative
Location	Official_Champ_Reflection_Token-9-13-25.sol#L272
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 than expected.

```
Shell

function claimReflections(address _user) public nonReentrant
onlyEOA {
    require(
    block.timestamp >= lastClaimTime[_user] + 30,
    "Wait 30s cooldown"
    );
    uint256 owed = calculateReflectionClaimable(_user);
    require(owed > 0, "No reflections to claim");
    lastClaimTime[_user] = block.timestamp;
    claimedEthReflections[_user] += owed;
    totalClaimedEthReflections += owed;
    (bool success, ) = payable(_user).call{value: owed}("");
    require(success, "ETH transfer failed");
    emit ReflectionsClaimed(_user, owed);
}
```

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	Official_Champ_Reflection_Token-9-13-25.sol#L272
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 flash loan. 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.

```Shell
function calculateReflectionClaimable(
address holder
) internal view returns (uint256) {
    if (
    isExcludedFromReward[holder] ||
    balanceOf(holder) < minimumHoldingForReflection
    ) return 0;
    uint256 totalSupplyExcludingBurned = totalSupply() -
    (balanceOf(address(0)) + tokenomicsHoldings);
    uint256 reflectionShare = (balanceOf(holder) *
    totalEthReflectionBalance) / totalSupplyExcludingBurned;
    return reflectionShare - claimedEthReflections[holder];
}
```

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	Official_Champ_Reflection_Token-9-13-25.sol#L142,204
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.

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

Additionally, the variable `tokenomicsHoldings` is used in the calculation of claimable shares and functions as a form of locked balance. However, it does not represent an actual portion of the token supply and can be arbitrarily set by the contract owner. This may be misleading for users and the actual token amounts to be claimed, leading to confusion and inconsistencies.

```Shell
Shell
function changeHoldingAmount(uint256 _tokens) external
onlyOwner {
require(_tokens > 0, "Minimum token value must be higher than
0");
tokenomicsHoldings = _tokens;

}
```

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	Official_Champ_Reflection_Token-9-13-25.sol#77
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.

```shell
constructor(
address _uniswapRouter,
...
) ERC20("CHAMP", "CCG") Ownable(_msgSender()) {
...
uniswapRouter = IUniswapV2Router02(_uniswapRouter);
...
}
```

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.

PSU - Potential Subtraction Underflow

Criticality	Minor / Informative
Location	Official_Champ_Reflection_Token-9-13-25.sol#L307,325
Status	Unresolved

Description

The contract subtracts two values, the second value may be greater than the first value if the contract owner misuses the configuration. As a result, the subtraction may underflow and cause the execution to revert. In particular, the `reflectionShare` of a user varies with the balance and may decrease beyond the already claimed rewards. As a result, all attempts to claim will revert with no error. The case is similar for `totalSupplyExcludingBurned`.

```Shell
uint256 totalSupplyExcludingBurned = totalSupply() -
(balanceOf(address(0)) + tokenomicsHoldings);

return reflectionShare - claimedEthReflections[holder];
```

Recommendation

The team is advised to properly handle the code to avoid underflow subtractions and ensure the reliability and safety of the contract. The contract should ensure that the first value is always greater than the second value. It should add a sanity check in the setters of the variable or not allow executing the corresponding section if the condition is violated.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	Official_Champ_Reflection_Token-9-13-25.sol#L137
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.

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

RCS - Redundant Conditional Statements

Criticality	Minor / Informative
Location	Official_Champ_Reflection_Token-9-13-25.sol#L165,170
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`.

```Shell
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.

USCR - Unclaimable Smart Contract Rewards

Criticality	Minor / Informative
Location	Official_Champ_Reflection_Token-9-13-25.sol#L300
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 `reflectionShare`, 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. Additionally, if the `minimumHoldingForReflection` is set to a value greater than the token's supply no one will be able to get rewards.

```Shell
function calculateReflectionClaimable(
address holder
) internal view returns (uint256) {
    if (
    isExcludedFromReward[holder] ||
    balanceOf(holder) < minimumHoldingForReflection
    ) return 0;
    uint256 totalSupplyExcludingBurned = totalSupply() -
    (balanceOf(address(0)) + tokenomicsHoldings);
    uint256 reflectionShare = (balanceOf(holder) *
    totalEthReflectionBalance) / totalSupplyExcludingBurned;
    return reflectionShare - claimedEthReflections[holder];
}
```

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	Official_Champ_Reflection_Token-9-13-25.sol#L5,115,130,137,142,272,291,318,327,328
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.

```Shell
bool _applyTax
uint256 _tokens
address _user
uint256 _buyTax
uint256 _sellTax
uint256 _liquidityAllocation
uint256 _reflectionsAllocation
```

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	Official_Champ_Reflection_Token-9-13-25.sol#L139,144
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.

```Shell
swapThresholdAmount = _tokens
tokenomicsHoldings = _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	Official_Champ_Reflection_Token-9-13-25.sol#L175,181
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.

```Shell
uint256 liquidityHalf = liquidityTax / 2
uint256 ethForLiquidity = (wethOutFromSwap * liquidityHalf) /
tokensToSwap
```

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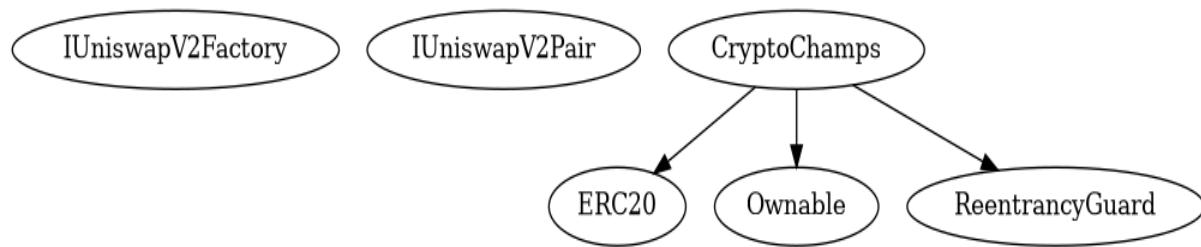
To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
CHAMP	Implementation	ERC20, Ownable, ReentrancyGuard		
		Public	✓	ERC20 Ownable
	changeApplyTax	External	✓	onlyOwner
	_excludeFromReward	Internal	✓	
	freezeExclusions	External	✓	onlyOwner
	changeMinimumHoldingForReflection	External	✓	onlyOwner
	changeSwapThreshold	External	✓	onlyOwner
	changeHoldingAmount	External	✓	onlyOwner
	_createLiquidityPool	Internal	✓	
	_update	Internal	✓	
	_swapTokensForWETH	Private	✓	nonReentrant
	_addToLiquidity	Private	✓	nonReentrant
	_distributeReflections	Internal	✓	
	claimReflections	Public	✓	nonReentrant onlyEOA
	getPendingReflections	External		-
	calculateReflectionClaimable	Internal		
	getClaimedEthAmount	External		-
	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. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions. There is also a limit of max 10% 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

cyberscope.io