Cyberscope

Audit Report

RAIFI Community

June 2025

SHA256

096631e23d7b2ee644bd6ca6ed78380786f6e3e422e02c44ec834358bbac9760

Audited by © cyberscope

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

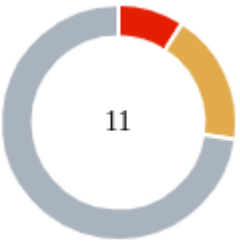
Audit Updates

Initial Audit	18 Jun 2025

Source Files

Filename	SHA256
Community.sol	096631e23d7b2ee644bd6ca6ed78380786f6e3e422e02c44ec834358bbac9760

Findings Breakdown



Critical	1
Medium	2
Minor / Informative	8

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	2	0	0	0
Minor / Informative	8	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	PUL	Potentially Unbounded Loop	Unresolved
●	FSR	Function Self Reference	Unresolved
●	IRR	Inconsistent Referral Removal	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	IAER	Inefficient Array Element Removal	Unresolved
●	MCRC	Missing Cyclic Reference Check	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved

PUL - Potentially Unbounded Loop

Criticality	Critical
Location	Community.sol#L55
Status	Unresolved

Description

The `_isReferrerInReferralChain` function includes a `depthLimit` variable inside a loop. This variable is declared and reset on every iteration, it could therefore lead to unbounded execution.

```
function _isReferrerInReferralChain(address _member, address
_potentialReferrer) internal view returns (bool) {
address current = members[_member].referrer;
while (current != address(0)) {
if (current == _potentialReferrer) {
return true;
}
current = members[current].referrer;
uint depthLimit = 100;
if (depthLimit-- == 0) break;
}
return false;
}
```

Recommendation

The team is advised to remove the `depthLimit` declaration from the `for` loop and only decrement it within the loop to enforce optimal traversal depth.

FSR - Function Self Reference

Criticality	Medium
Location	Community.sol#L124
Status	Unresolved

Description

The `_getFullReferralTreeStakeInternal` function recursively calls itself with addresses from the directReferrals array. If any of these addresses is the zero address, the function lacks a termination mechanism, potentially resulting in an unbounded recursive loop.

```
(uint subTreeCount, uint subTreeStake) =
_getFullReferralTreeStakeInternal(directReferrals[i]);
```

Recommendation

The team is advised to introduce an explicit check for address(0) before making the recursive call. This will ensure proper termination and protect against unintended execution paths.

IRR - Inconsistent Referral Removal

Criticality	Medium
Location	Community.sol#L82
Status	Unresolved

Description

The `updateReferrer` function attempts to remove msg.sender from the old referrer's referrals array by shifting elements and then calling `.pop()`. However, the current implementation removes the last element of the array unconditionally after the shift, which may not correspond to msg.sender. This can lead to the unintended removal of an unrelated third party from the referral list.

```
for (uint j = i; j < oldReferrals.length - 1; j++) {
members[oldReferrer].referrals[j] = oldReferrals[j + 1];
}
 members[oldReferrer].referrals.pop();
```

Recommendation

The team is advised to ensure that only the intended element is removed to maintain referral integrity.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	Community.sol#L29
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.

```
owner
```

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.

IAER - Inefficient Array Element Removal

Criticality	Minor / Informative
Location	Community.sol#L79
Status	Unresolved

Description

The contract utilizes a method for removing elements from an array. Specifically, the function employs a for loop to iterate through the array elements, shifting each element down by one index to remove the specified element. This approach, while functional, could be more optimal in terms of gas usage and execution time, especially as the size of the array grows.

```
for (uint j = i; j < oldReferrals.length - 1; j++) {
members[oldReferrer].referrals[j] = oldReferrals[j + 1];
}
```

Recommendation

It is recommended to enhance the efficiency of the function by adopting a more gas-efficient approach. This can be achieved by swapping the last element of the array with the element intended for removal, and then calling the `pop` method to remove the last element. This method significantly reduces the number of operations required, especially for large arrays, optimizing gas costs and execution time.

MCRC - Missing Cyclic Reference Check

Criticality	Minor / Informative
Location	Community.sol#L68
Status	Unresolved

Description

The `updateReferrer` function allows existing members to assign a new referrer but does not perform a check for cyclic references. This enables users to create circular referral structures, which can lead to code inconsistencies.

```
function updateReferrer(address _newReferrer) external {
require(isMember[msg.sender], "Not a member yet.");
require(msg.sender != _newReferrer, "Cannot refer yourself.");
address oldReferrer = members[msg.sender].referrer;
...
}
```

Recommendation

The team is advised to implement the existing cyclic reference check in `updateReferrer` to maintain consistency and prevent referral graph corruption.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	Community.sol#L33
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
function setStakingContract(address _stakingContract) external onlyOwner {
require(_stakingContract != address(0), "Staking contract address cannot be
zero.");
stakingContract = _stakingContract;
staking= IStakingRAI(stakingContract);
}
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	Community.sol#L33,39,68,137
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.

```
address _stakingContract
address _referrer
address _newReferrer
uint256 _level
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	Community.sol#L44
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
MemberInfo memory newMember
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	Community.sol#L30
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.

```
stakingContract = _stakingContract
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	Community.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 ^0.8.20;
```

Recommendation

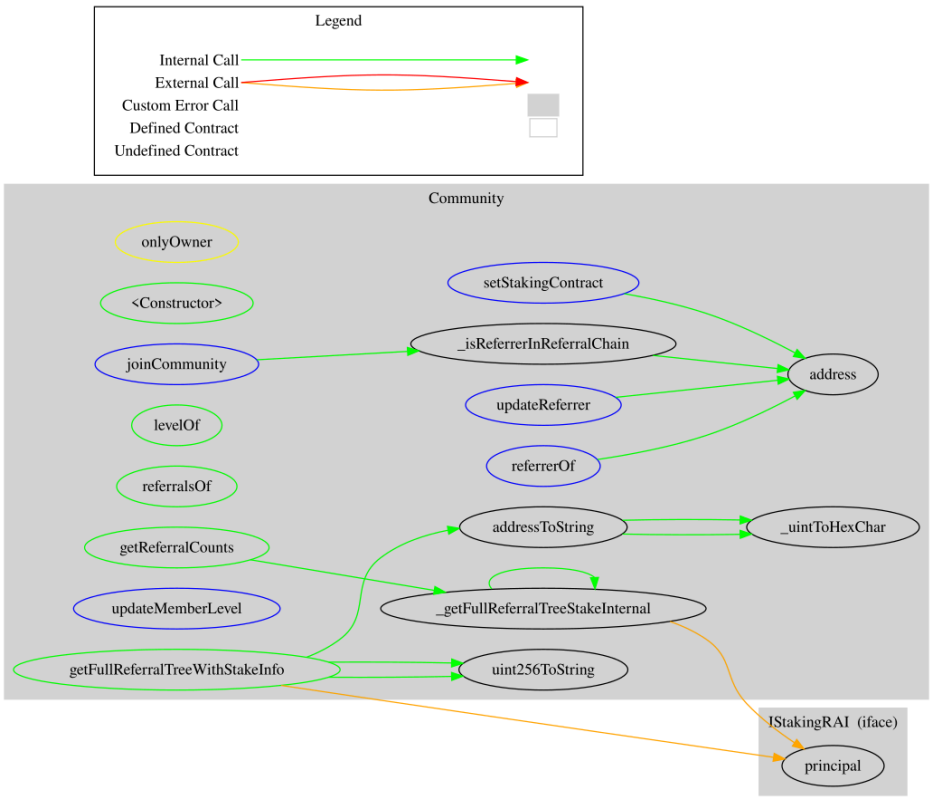
The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest 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**
IStakingRAI	Interface			
	principal	External		-
Community	Implementation			
		Public	✓	-
	setStakingContract	External	✓	onlyOwner
	joinCommunity	External	✓	-
	_isReferrerInReferralChain	Internal		
	updateReferrer	External	✓	-
	levelOf	Public		-
	referrerOf	External		-
	referralsOf	Public		-
	getReferralCounts	Public		-
	_getFullReferralTreeStakeInternal	Internal		
	updateMemberLevel	External	✓	onlyOwner
	getFullReferralTreeWithStakeInfo	Public		-
	addressToString	Internal		
	_uintToHexChar	Internal		
	uint256ToString	Internal		

Inheritance Graph

Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
Community
onlyOwner
<Constructor>
joinCommunity
levelOf
referralsOf
getReferralCounts
updateMemberLevel
getFullReferralTreeWithStakeInfo
setStakingContract
_isReferrerInReferralChain
updateReferrer
referrerOf
addressToString
_getFullReferralTreeStakeInternal
uint256ToString
address
_uintToHexChar
IStakingRAI (iface)
principal

Summary

RAIFI contract implements a referral mechanism. This audit investigates security issues, business logic concerns and potential improvements.

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