Cyberscope

Audit Report

ProfitScraper

March 2023

Network	BSC
Address	0x40DA8F4e3922b74F36B19e4DdBC1d623e8EFBD98
Audited by	© cyberscope

Table of Contents

Table of Contents 1
Review 3
Audit Updates 3
Source Files 3
Introduction 4
Roles 4
Owner 4
User 4
Diagnostics 5
UC - Unlimited Claims 7
Description 7
Recommendation 7
MC - Missing Check 8
Description 8
Recommendation 8
MMN - Misleading Method Naming 9
Description 9
Recommendation 9
PTRP - Potential Transfer Revert Propagation 10
Description 10
Recommendation 10
PRC - Possible Redundant Calculation 11
Description 11
Recommendation 11
CR - Code Repetition 12
Description 12
Recommendation 12
PRE - Potential Reentrance Exploit 13
Description 13
Recommendation 13
MVW - Multiple Variable Writes 14
Description 14
Recommendation 14
TUU - Time Units Usage 15
Description 15
Recommendation 15
PSU - Potential Subtraction Underflow 16
Description 16

Recommendation 16
RSML - Redundant SafeMath Library 17
Description 17
Recommendation 17
IDI - Immutable Declaration Improvement 18
Description 18
Recommendation 18
L02 - State Variables could be Declared Constant 19
Description 19
Recommendation 19
L04 - Conformance to Solidity Naming Conventions 20
Description 20
Recommendation 21
L07 - Missing Events Arithmetic 22
Description 22
Recommendation 22
L13 - Divide before Multiply Operation 23
Description 23
Recommendation 23
L15 - Local Scope Variable Shadowing 24
Description 24
Recommendation 24
Functions Analysis 25
Inheritance Graph 27
Flow Graph 28
Summary 29
Disclaimer 30
About Cyberscope 31

Review

Explorer	https://bscscan.com/address/0x40da8f4e3922b74f36b19e4ddbc1d623e8efbd98
Address	0x40DA8F4e3922b74F36B19e4DdBC1d623e8EFBD98

Audit Updates

Initial Audit	22 Mar 2023

Source Files

Filename	SHA256
ProfitScraper.sol	1223f25a1502e81abf28a7a60250a880dbdb91ffbcfb0e6db7c093dfa1d5b189

Introduction

The ProfitScraper contract allows users to deposit ether and earn rewards over time. It also includes referral rewards for bringing in new users. The contract owner takes a fee from each deposit and withdrawal. The contract keeps track of various statistics such as total invested, total rewards, and total investors. The contract includes safety measures to prevent reentrancy attacks.

Roles

Owner

The onwer has authority over the following functions:

- `function resetContract(address _devWallet)`
- `function getAmount(uint _amount)`
- `function updateContract()`

User

The user can interact with the following functions:

- `function _getNextDepositID()`
- `function deposit(address _referrer)`
- `function claimAllReward()`
- `function getOwnedDeposits(address investor)`
- `function getAllClaimableReward(address _investor)`
- `function getBalance()`
- `function getTotalRewards()`
- `function getTotalInvests()`
- `function getTotalAmountEarned(address _investor)`

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	UC	Unlimited Claims	Unresolved
●	MC	Missing Check	Unresolved
●	MMN	Misleading Method Naming	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	PRC	Possible Redundant Calculation	Unresolved
●	CR	Code Repetition	Unresolved
●	PRE	Potential Reentrance Exploit	Unresolved
●	MVW	Multiple Variable Writes	Unresolved
●	TUU	Time Units Usage	Unresolved
●	PSU	Potential Subtraction Underflow	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved

●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved

UC - Unlimited Claims

Criticality	Critical
Location	ProfitScraper.sol#L256
Status	Unresolved

Description

The user has the ability to claim rewards indefinitely. The user may take advantage of this by calling the `claimAllReward` function multiple times. The contract implements no restriction mechanism to prevent users from exploiting the function's vulnerability.

```
function claimAllReward() public nonReentrant {}
```

Recommendation

The team is advised to add constraints to the `claimAllReward` function so that users will not be able to abuse it.

MC - Missing Check

Criticality	Minor / Informative
Location	ProfitScraper.sol#L212
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.

The `investors[_referrer].referrer` can potentially be the zero address. As a result, the zero address will be added as a key to the `investors` mapping.

```
investors[investors[_referrer].referrer].referralsTwoLvlNumber += 1;
```

Recommendation

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

MMN - Misleading Method Naming

Criticality	Minor / Informative
Location	ProfitScraper.sol#L159
Status	Unresolved

Description

Methods can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some method names that are too generic or do not clearly convey the underneath functionality. Misleading method names can lead to confusion, making the code more difficult to read and understand.

The method `resetContract` sets a new value to the `devWallet` variable. As a result, the method name is inaccurate and misleading.

```
function resetContract(address _devWallet) public onlyOwner {
    require(_devWallet != address(0),"Please provide a valid address");
    devWallet = _devWallet;
}
```

Recommendation

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

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	ProfitScraper.sol#L186
Status	Unresolved

Description

The contract sends funds to a `devWallet` as part of the deposit 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 contract and revert the transfer.

```
payable(devWallet).transfer(depositFee);
```

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 achived by not allowing set contract addresses or by sending the funds in a non-revertable way.

PRC - Possible Redundant Calculation

Criticality	Minor / Informative
Location	ProfitScraper.sol#L218
Status	Unresolved

Description

The contract calculates the `referrerAmountlvl2` variable which is only used inside a conditional statement. The statement may or may not resolve to true. If it does not, then the calculation will be redundant, as this value is not being used elsewhere in the contract.

```
uint referrerAmountlvl2 = (_amount * REFFER_REVARD_2_LVL).div(PERCENT_RATE);
...
if(investors[_referrer].referrer != address(0)) {
    investors[investors[_referrer].referrer].referAmount =
investors[investors[_referrer].referrer].referAmount.add(referrerAmountlvl2);

    payable(investors[_referrer].referrer).transfer(referrerAmountlvl2);
}
```

Recommendation

The team is advised to calculate the `referrerAmountlvl2` variable inside the conditional statement. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

CR - Code Repetition

Criticality	Minor / Informative
Location	ProfitScraper.sol#L239,264,354,366
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

The contract calculates the claimable reward of the user in multiple instances, and then adds the new value to the `claimableAmount` variable. As a result, the same code is exists more than once.

```
uint lastRoiTime = block.timestamp - investors[_owner].lastCalculationDate;
uint allClaimableAmount = (lastRoiTime * investors[_owner].totalLocked *
investors[_owner].APR).div(PERCENT_RATE * REWARD_PERIOD);
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the `getAllClaimableReward` function, that already exists in the contract, and assign the returning value of the function to the `claimableAmount` variable. That way, the contract will avoid repeating the same code in multiple places.

PRE - Potential Reentrance Exploit

Criticality	Minor / Informative
Location	ProfitScraper.sol#L173
Status	Unresolved

Description

As part of the `deposit` method, the contract transfers ETH to the referrer if the referrer's address is not equal to the zero address. Since the referrer can be any address, the address can be exploited for a re-entrance attack.

```
function deposit(address _referrer) public payable {
    ...
    payable(investors[msg.sender].referrer).transfer(referrerAmountlvl1);
    ...
    payable(investors[_referrer].referrer).transfer(referrerAmountlvl2);
    ...
}
```

Recommendation

The team is advised to prevent the re-entrance exploit as part of the solidity best practices. Some suggestions are:

- Not allow contract addresses to receive funds.
- Add a locker/mutex in the transfer method scope.
- Transfer the funds as the last statement of the transfer method, so that the balance will have been subtracted during the re-entrance phase.

MVW - Multiple Variable Writes

Criticality	Minor / Informative
Location	ProfitScraper.sol#L199,247
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.

During the deposit flow, the contract assigns the same value to the `lastCalculationDate` property twice. Writing to storage multiple times can increase the gas cost drastically.

```
investors[msg.sender].lastCalculationDate = block.timestamp;
```

Recommendation

The team is advised to take these segments into consideration 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 executing it. A recommended approach would be to remove the first assignment and only keep the last one.

TUU - Time Units Usage

Criticality	Minor / Informative
Location	ProfitScraper.sol#L344
Status	Unresolved

Description

The contract is using arbitrary numbers to form time-related values. As a result, it decreases the readability of the codebase and prevents the compiler to optimize the source code.

```
uint add5 = ((block.timestamp - investors[_owner].lastCalculationDate) / 60 * 60 *
24);
```

Recommendation

It is a good practice to use the time units reserved keywords like `seconds`, `minutes`, `hours`, `days`, `weeks` and `years` to process time-related calculations.

It's important to note that these time units are simply a shorthand notation for representing time in seconds, and do not have any effect on the actual passage of time or the execution of the contract. The time units are simply a convenience for expressing time in a more human-readable form.

PSU - Potential Subtraction Underflow

Criticality	Minor / Informative
Location	ProfitScraper.sol#L260
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.

The `changeAPR` variable is calculated by multiplying the number of days a user has not claimed the reward by 5. The initial `APR` value is 510. This means that if the user claims the reward after 102 days or more then the operation at code segment below will lead to an underflow.

```
investors[msg.sender].APR -= changeAPR;
```

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.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	ProfitScraper.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert on underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases the gas consumption unnecessarily.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	ProfitScraper.sol#L156
Status	Unresolved

Description

The contract is using variables that initialize them only in the constructor. The other functions are not mutating the variables. These variables are not defined as `immutable`.

```
deploymentDate
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	ProfitScraper.sol#L338
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.

The variable `bnb100` is declared inside the `updateContract` function and its value never changes. Declaring and initializing the same variable every time inside the function is redundant and consumes more gas.

```
uint bnb100 = 100 * 10 ** 18;
```

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, as well as, move them to global scope so that they are initialized only once.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	ProfitScraper.sol#L159,173,292,303,329
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 _devWallet
address _referrer
uint _amount
address _investor
```

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/v0.8.17/style-guide.html#naming-convention.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	ProfitScraper.sol#L249
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.

```
totalInvested = totalInvested.add(_amount)
```

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	ProfitScraper.sol#L233,236,344
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.

```
uint percent = ((investors[msg.sender].totalLocked + _depositAmount) / (10 * 10 **
18))
investors[msg.sender].APR += percent * 50
```

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.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	ProfitScraper.sol#L343
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
address _owner = firstDeposits[i].owner
```

Recommendation

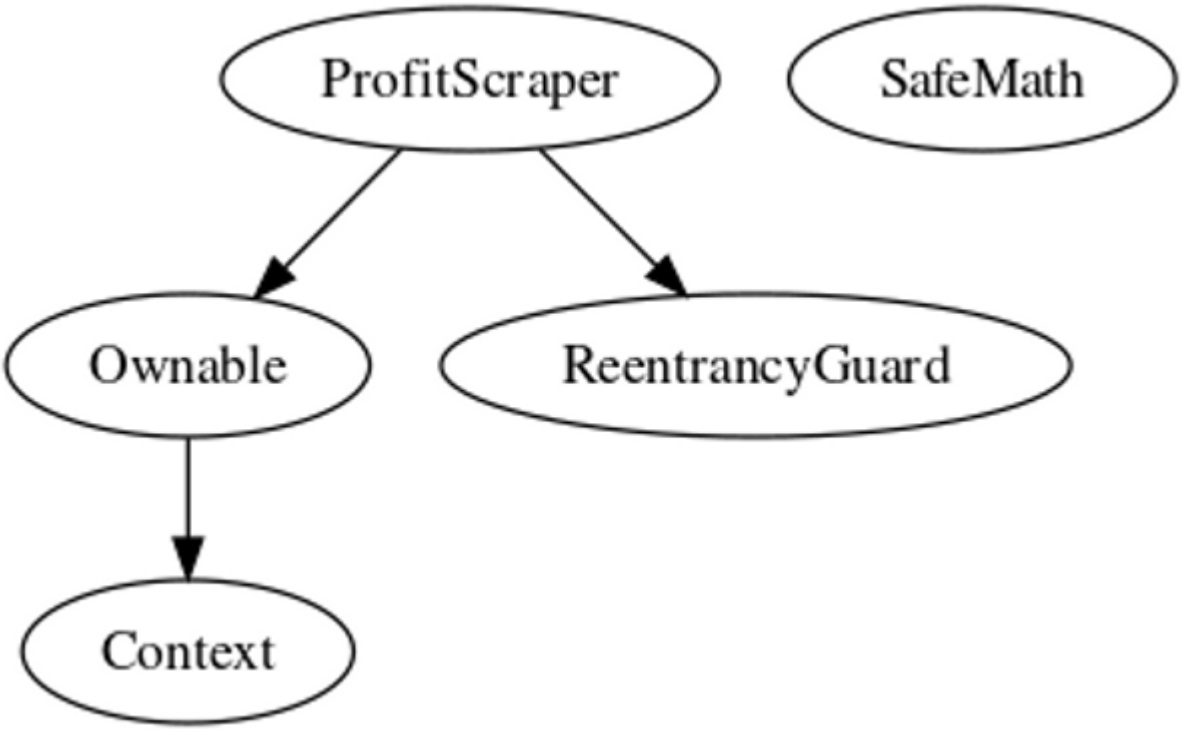
It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

Functions Analysis

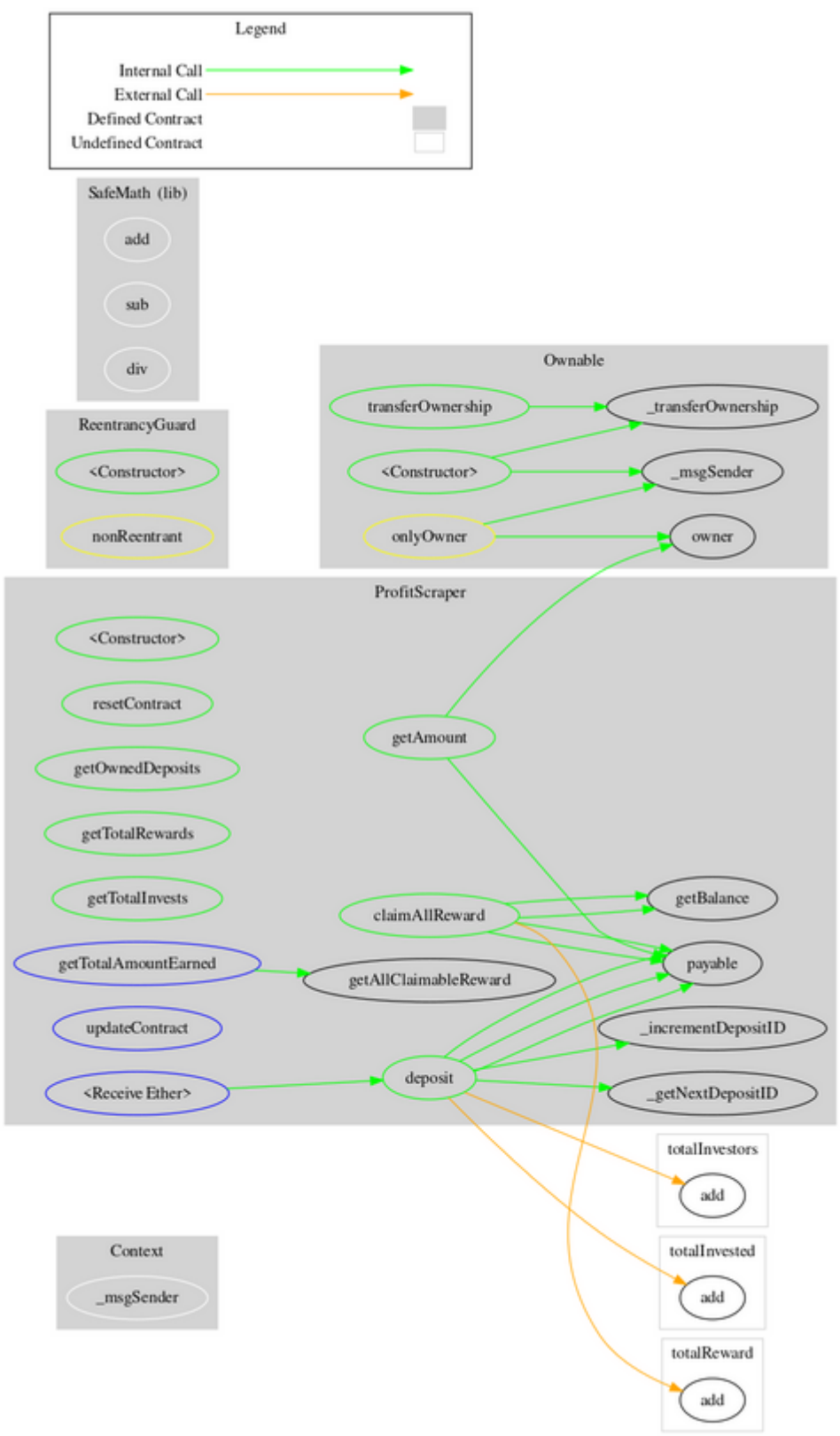
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
ReentrancyGuard	Implementation			
		Public	✓	-
SafeMath	Library			
	add	Internal		
	sub	Internal		
	div	Internal		
ProfitScraper	Implementation	Ownable, ReentrancyGuard		
		Public	✓	-
	resetContract	Public	✓	onlyOwner
	_getNextDepositID	Private		
	_incrementDepositID	Private	✓	
	deposit	Public	Payable	-

	claimAllReward	Public	✓	nonReentrant
	getAmount	Public	Payable	onlyOwner
	getOwnedDeposits	Public		-
	getAllClaimableReward	Public		-
	getBalance	Public		-
	getTotalRewards	Public		-
	getTotalInvests	Public		-
	getTotalAmountEarned	External		-
	updateContract	External	✓	onlyOwner
		External	Payable	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
SafeMath (lib)
add
sub
div
ReentrancyGuard
<Constructor>
nonReentrant
Ownable
transferOwnership
_transferOwnership
<Constructor>
_msgSender
onlyOwner
owner
ProfitScraper
<Constructor>
resetContract
getOwnedDeposits
getTotalRewards
getTotalInvests
getTotalAmountEarned
updateContract
<Receive Ether>
getAmount
claimAllReward
getAllClaimableReward
deposit
getBalance
payable
_incrementDepositID
_getNextDepositID
totalInvestors
add
totalInvested
add
totalReward
add
Context
_msgSender

Summary

ProfitScraper contract implements a staking and rewards 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

https://www.cyberscope.io