Cyberscope

Audit Report

Ratpad

April 2023

SHA256 bd9792d736ec3e83b1098339a433bec59755cd49dbc53dea54a06ca28e98cee4

Audited by © cyberscope

Table of Contents

Review

Explorer	https://testnet.bscscan.com/address/0xf896acea879b8c5088ec10ca8210697c97b051b0

Audit Updates

Initial Audit	22 Mar 2023 https://github.com/cyberscope-io/audits/blob/main/ratpad/v1/audit.pdf
Corrected Phase 2	07 Apr 2023 ttps://github.com/cyberscope-io/audits/blob/main/ratpad/v2/audit.pdf
Corrected Phase 3	13 Apr 2023

Source Files

Filename	SHA256
LaunchPad.sol	bd9792d736ec3e83b1098339a433bec59755cd49dbc53dea54a06ca28e98cee4

Findings Breakdown

Critical	2
Medium	0
Minor / Informative	18

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	2	0	0
Medium	0	0	0	0
Minor / Informative	14	4	0	0

Introduction

Cyberscope audited three contracts that include a Presale Launcher Master contract, a Token Launcher Master Contract, and a Lock Launcher Master Contract. These contracts are designed to deploy child contracts dynamically, which include a Presale child contract called PresaleContract, a Token child contract called ERC20, and a Lock child contract called tokenLock.

PresaleContract

The Launchpad contract serves as a factory contract for presales and is responsible for maintaining the essential registries related to presale contracts. In addition, the contract handles the configuration of fees, presale tiers and the fee receiver.

Roles

The contract roles consist of the `admin` roles. The `admin` is responsible for the configuration of fees and the fee receiver.

The users have the authority to:

- Create a new presale.
- Get fees in relation to the user's tier.
- Set lock contract.
- Get pool details.
- Get tier in relation to the address.
- Change contract tiers.

PresaleContract

The contract is a PresaleContract, which handles the presale of tokens before they are released to the public. The presale allows investors to buy tokens before they are available for purchase on a public exchange.

Roles

The contract roles consist of the `admin` and the presale `owner` role.

The `admin` is responsible for configuring badges and admin investing authorization.

The presale `owner` is responsible for editing the pool if it is allowed and finalizing or canceling the presale.

The users have the authority to:

- Claim bought token when presale finalizes.
- View bought tokens.
- Withdraw bought token with fee.
- Buy presale tokens.
- View contribution array.
- Add liquidity to the token.
- View presale badges
- View presale info.
- View if the presale is canceled or finished.

TokenLauncher

The tokenLauncher contract has several functions and events to facilitate the dynamic creation and management of ERC20 tokens.

Roles

The contract roles consist of the `admin` role. The `admin` has the authority to configure fees and change admin.

The `users` have the authority to:

- Create a new token.
- View created tokens.
- View created tokens created by a specified address.

TokenLockLauncher

The tokenLockLauncher contract is a Locker factory contract that enables the creation and management of locked ERC20 tokens.

Roles

The contract roles consist of the `admin` and the `Launcher`.

The `users` have the authority to:

- Lock token.
- View all locked tokens.

TokenLock

The `TokenLock` contract implements a simple locker. Where the contract keeps the tokens locker until the lock time elapses.

Roles

The contract roles consist of the `owner` role. The owner has the authority to Withdraw locked tokens when the lock times elapse.

The `users` can view the Locker data.

Contract Quality

The audit scope is to check for security vulnerabilities, validate the business logic and propose potential optimizations. The contract is missing the fundamental principles of a Solidity smart contract regarding gas consumption, code readability, and data structures. Given these issues, it is not advisable to assume that the contract is in a production-ready state. The development team is strongly encouraged to re-evaluate the business logic and Solidity guidelines to ensure that the contract adheres to established best practices and security measures. It is recommended that the team review the contract's vulnerabilities and optimize it accordingly to minimize the security issues and improve the contract's efficiency. The code's readability should also be improved by simplifying function definitions and using descriptive variable names, as this will enhance the contract's auditability and maintenance.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	PTAI	Potential Transfer Amount Inconsistency	Acknowledged
●	MSC	Missing Sanity Checks	Acknowledged
●	OLD	Override Lock Data	Acknowledged
●	CR	Code Readability	Unresolved
●	DSO	Data Structure Optimization	Acknowledged
●	LDAR	Large Data Array Return	Acknowledged
●	RAV	Redundant Array Variable	Acknowledged
●	DD	Data Duplication	Unresolved
●	RDM	Revert Descriptive Message	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L06	Missing Events Access Control	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved

●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L18	Multiple Pragma Directives	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

PTAI - Potential Transfer Amount Inconsistency

Criticality	Critical
Location	LaunchPad.sol#L1029,1034,1246,1343,1361,1549,1562
Status	Acknowledged

Description

The `transfer()` and `transferFrom()` functions are used to transfer a specified amount of tokens to an address. The fee or tax is an amount that is charged to the sender of an ERC20 token when tokens are transferred to another address. According to the specification, the transferred amount could potentially be less than the expected amount. This may produce inconsistency between the expected and the actual behavior.

The following example depicts the diversion between the expected and actual amount.

Tax	Amount	Expected	Actual
No Tax	100	100	100
10% Tax	100	100	90

```
Token.safeTransfer(msg.sender, entitlement);
Token.transfer(
    selfInfo._token_owner_admin_currency[1],
    Token.balanceOf(address(this))
);
_Token.safeTransferFrom(
    msg.sender,
    address(tx1),
    (integers[0]+integers[18])
);
Token.safeTransferFrom(msg.sender, address(tx1), _amount);
_token.transfer(Data.withdrawer,
_token.balanceOf(address(this)));
```

Recommendation

The team is advised to take into consideration the actual amount that has been transferred instead of the expected.

It is important to note that an ERC20 transfer tax is not a standard feature of the ERC20 specification, and it is not universally implemented by all ERC20 contracts. Therefore, the contract could produce the actual amount by calculating the difference between the transfer call.

```
Actual Transferred Amount = Balance After Transfer - Balance Before Transfer
```

Team Update

"*This is not doable and our assumption is that the user will make the presale contract exempt from taxes.*"

MSC - Missing Sanity Checks

Criticality	Critical
Location	LaunchPad.sol#961,982,1540,1558,994,992,1444
Status	Acknowledged

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 variables `feeOnFinalization` , `withdrawFee` , and fee are not properly sanitized. The values could exceed the denominators' values and potentially create unwanted behavior.

```
function changeFeeForPoolCreation(
    uint256 _feeFinalization,
    uint256 tier1Fee,
    uint256 tier2fee,
    uint256 tier3fee,
    uint256 _withdrawFee
) public onlyAdmin {
    require(_feeFinalization>0);
    require(tier1Fee>0);
    require(tier2fee>0);
    require(tier3fee>0);
    require(_withdrawFee>0);
    feeOnFinalization = _feeFinalization;
    feeForPoolTier1 = tier1Fee;
    feeForPoolTier2 = tier2fee;
    feeForPoolTier3 = tier3fee;
    withdrawFee = _withdrawFee;
}
```

The variable `feeReceiverAddress` is not properly sanitized. It could be set to zero address.

```
//contract/Launchpad
function changeReceiver(address _receiver) public onlyAdmin {
    feeReceiverAddress = _receiver;
}
```

The `time` variable is not properly sanitized.

```
//contract/tokenLockLauncher
tokenLock tx1 = new tokenLock(
    ...
    _time,
    block.timestamp,
    ...
);
```

The `min` and `max` , `start` and `end` and the time variables of the `integers` array are not properly sanitized.

The addresses of the `addresses` are not properly sanitized.

```
//contract/Presale-Launchpad
address[] addresses;
uint256[] integers;
```

The arguments `min1` and `min2` are not properly sanitized.

```
function changeTiers(uint256 min1, uint256 min2) public {
    tier1ratboyMin = min1;
    tier2ratboyMin = min2;
}
```

Recommendation

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

- The fee nominators should be lower than 100.
- All the addresses should not be set to zero addresses.
- The `time` variable should be greater than the current timestamp.
- The contract should examine and properly sanitize the tier configuration.

Team Update

"*These two variables are being sent from the launchpad contract. Users are not sending it directly to presale contract.*"

OLD - Override Lock Data

Criticality	Minor / Informative
Location	LaunchPad.sol#L991
Status	Acknowledged

Description

The Lock data can be overridden after a presale is completed. If someone created a presale with the same address, the lock data will be overridden. This could lead to unintended behavior, potential security vulnerabilities, and unfair treatment of investors who participated in the presale.

```
function createPresale(
    address[] memory addresses,
    string[] memory strings,
    uint256[] memory integers,
    string memory _hash
) public payable {
    require(msg.value >= getFee(msg.sender),"Less fee is provided");
    if(tokenPresale[addresses[0]] !=address(0)){
        PresaleContract tx3 = PresaleContract(payable(tokenPresale[addresses[0]]));
        bool adminAllowed = tx3.getSelfInfo().adminAllowed;
        require(!adminAllowed);
    }
    ...
    TokenPresale[_token_owner_admin_currency[0]] = address(tx1);
    TokenPresaleLocked[_token_owner_admin_currency[0]][
    ...
}
```

Recommendation

It is recommended updating the presale function to ensure that it cannot be override old data. This can be achieved by adding a condition that checks whether the presale has ended before and whether the same token address is used. Additionally, we recommend thoroughly testing the updated function to verify that it functions as intended and does not introduce any new issues.

Team Update

"*That issue is already sorted, handling so that no presale can be created if one presale for token is approved or finalized , The override is possible only if new presale is created on same token address.. so we handled it so that no presale can be created if one presale for token is approved or finalized*"

CR - Code Readability

Criticality	Minor / Informative
Location	LaunchPad.sol#L936
Status	Unresolved

Description

There are array variables that contain multiple different variables of different contexts. This can result in difficulties to understand the purpose of these variables and how they're used within the contract.

When there are multiple array variables that contain multiple different variables of different contexts, it can be difficult to discern the purpose and usage of each variable. This can increase the likelihood of errors and omissions during the audit, potentially leading to security vulnerabilities and other issues.

```
address[] addresses;
string[] strings;
uint256[] integers;
```

Recommendation

To improve the readability and clarity of the Solidity contract, it's recommended to use different variables and group variables with a similar context only. By doing so, auditors and developers can better understand the purpose and usage of each variable, reducing the likelihood of errors and omissions during the audit.

In addition, it's recommended to use clear and descriptive variable names, making it easier to understand the purpose and usage of each variable. By using descriptive variable names, developers can help to ensure that the Solidity contract is more readable and easier to audit.

DSO - Data Structure Optimization

Criticality	Minor / Informative
Location	LaunchPad.sol#L1158,1411,1415
Status	Acknowledged

Description

The contract utilizes the `Badges` variable as an array, but the business logic of the contract does not require the array to be dynamic. This results in unnecessary memory usage, slower execution times, and more operations being performed than necessary. Therefore, optimizing this code segment could help reduce these issues.

```
bool public Badges;

function addBadges(bool[] memory _badges) public onlyAdmin {
    Badges = _badges;
}

function getBadges() public view returns (bool[] memory) {
    return Badges;
}
```

Recommendation

The contract could use a fixed data structure. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

Team Update

"We kept it dynamic because we didnt want to restrict ourselves."

LDAR - Large Data Array Return

Criticality	Minor / Informative
Location	LaunchPad.sol#L1073
Status	Acknowledged

Description

The contract contains a function that returns a large array of data, which could make it difficult to use in practice. If this function is called frequently, it could cause performance issues.

The array `PresaleArray` , `arr1` might return a large array of data.

```
function getPoolDetails()
  public
  view
  returns (Presale[] memory, PresaleSubData[] memory)
{
  PresaleSubData[] memory arr1 = new
PresaleSubData[](presaleArray.length);
  for (uint256 i = 0; i < presaleArray.length; i++) {
      PresaleContract tx1 =
PresaleContract(payable(presaleArray[i]._address));
      (
          ,
          ,
          uint256 purchasedTokens,
          uint256 investedAmount,
          uint256 finalized,
          address LP,
          bool adminAllowed
      ) = tx1.selfInfo();
      PresaleSubData memory tx2 = PresaleSubData(
          investedAmount,
          purchasedTokens,
          finalized,
          LP,
          adminAllowed
      );
      arr1[i] = tx2;
  }
  return (presaleArray, arr1);
}
```

Recommendation

It is recommended to consider implementing pagination to break the large array into smaller, more manageable chunks. This approach can reduce the computational burden on the function and make it easier to use in practice. By implementing pagination, the contract can improve its usability and overall performance.

RAV - Redundant Array Variable

Criticality	Minor / Informative
Location	LaunchPad.sol#L939
Status	Acknowledged

Description

The contract contains an array with two variablea that are redundant and not used in any of the contract's logic. The indexes 2 and 3 of the `addresses` array are never used in the contract.

```
address[] addresses;
```

Recommendation

It is recommended to remove the redundant variable from the contract to simplify the codebase and reduce unnecessary gas costs. Before removing the variable, it is important to ensure that it is not used anywhere else in the contract, including any external dependencies. By removing the redundant variable, the contract can become more streamlined and efficient, potentially reducing the chances of errors and making it easier to maintain and audit in the future.

Team Update

"In order to reduce the contract size, we store the value of this array in state variable and then used as and when required."

DD - Data Duplication

Criticality	Minor / Informative
Location	LaunchPad.sol#L1479
Status	Unresolved

Description

The TopCarde contract duplicates `WKDNFTToken` variables. This can result in unnecessary code complexity, potential inconsistencies in the data, and difficulty in maintaining the codebase.

```
ERC20 tx1 = new ERC20(
    _name,
    _symbol,
    _decimals,
    _totalSupply,
    msg.sender
);
TokenStruct memory tx2 = TokenStruct(
    address(tx1),
    _name,
    _symbol,
    _decimals
);
```

The Launchpad contract duplicates `Presale` data on mapping. All the necessary data is already saved on the presale contract. This can result in unnecessary code complexity, potential inconsistencies in the data, and difficulty in maintaining the codebase.

```
PresaleContract tx1 = new PresaleContract(
    addresses,
    strings,
    integers,
    _hash,

    feeOnFinalization,
    feeReceiverAddress,
    admin
);

Presale memory tx2 = Presale(
    presaleIndex,
    address(tx1),
    addresses,
    strings,
    integers,
    _hash,
    0,
    0
);
```

Recommendation

To prevent the duplication of values in multiple contracts, one option is to directly access the variables from the other contract. Another approach is to create a separate contract that contains the shared data or variables, which can then be inherited by other contracts that need access to them.

It is recommended to remove redundant data structures.

RDM - Revert Descriptive Message

Criticality	Minor / Informative
Location	LaunchPad.sol
Status	Unresolved

Description

The `revert` and `require` statements are used to halt the execution of a contract and revert any changes made to the contract's state. The contract does not provide a descriptive message to the `revert` and `require` functions.

```
require(msg.value >= selfInfo.integers[3] && msg.value <=
selfInfo.integers[2]||selfInfo.integers[5] <= block.timestamp);
require(msg.sender == selfInfo.integers[1]);
...
```

Recommendation

The team is suggested to provide a descriptive message to the `revert` and `require` function. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	LaunchPad.sol#L915,1153
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.

```
uint256 minSuperRat = 1
bool public publicSale = false
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	LaunchPad.sol#L427,840,913,919,958,962,978,991,1039,1098,1127,1164,1173,1220,1226,1265,1407,1441,1455,1460,1467,1468,1469,1470,1493,1511,1513,1537,1538,1539,1540,1541,1542,1543,1586,1602
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.

```
uint8 private Decimals
function WETH() external pure returns (address);
TokenLockLauncher public LockLauncher
event presaleCreated(address indexed presaleAddress);
uint256 _feeFinalization
uint256 _withdrawFee
address _receiver
address _admin
string memory _hash
address _user
uint _totalRecords
uint _start
IUniswapV2Router02 public Router

...
```

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.

L06 - Missing Events Access Control

Criticality	Minor / Informative
Location	LaunchPad.sol#L982
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. There are functions that have no event emitted, so it is difficult to track off-chain changes.

```
admin = _admin
```

Recommendation

To avoid this issue, 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.

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	LaunchPad.sol#L970,1141,1457
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.

```
feeForPoolTier1 = tier1Fee
tier1RatboyMin = min1
fee = _fee
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	LaunchPad.sol#L91,351
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function functionCall(address target, bytes memory data)
internal returns (bytes memory) {
        return functionCallWithValue(target, data, 0, "Address:
low-level call failed");
    }

function _callOptionalReturnBool(IERC20 token, bytes memory
data) private returns (bool) {
        // We need to perform a low level call here, to bypass
Solidity's return data size checking mechanism, since
        // we're implementing it ourselves. We cannot use
{Address-functionCall} here since this should return false
        // and not revert is the subcall reverts.

        (bool success, bytes memory returndata) =
address(token).call(data);
        return
            success && (returndata.length == 0 ||
abi.decode(returndata, (bool))) &&
Address.isContract(address(token));
    }
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	LaunchPad.sol#L1248,1253,1274,1280,1335,1337
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 grossEntitlement = (userEntitlement[msg.sender] *
selfInfo.integers[14] * cycleElapsed) / 100
uint256 cycleElapsed = (block.timestamp - selfInfo.finalized) /
                selfInfo.integers[4] <=
                (selfInfo.integers[19])
                ? (block.timestamp - selfInfo.finalized) /
selfInfo.integers[4]
                :  (selfInfo.integers[19])
```

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	LaunchPad.sol#L1213,1217,1462,1463,1532
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.

```
admin = _admin
feeReceiver = _receiver
feeReceiver = receiver
admin = _user
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	LaunchPad.sol#L238
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {
                let returndata_size := mload(returndata)
                revert(add(32, returndata), returndata_size)
            }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	LaunchPad.sol#L9,376,416
Status	Unresolved

Description

If the contract includes multiple conflicting pragma directives, it may produce unexpected errors. To avoid this, it's important to include the correct pragma directive at the top of the contract and to ensure that it is the only pragma directive included in the contract.

```
pragma solidity 0.8.18;
pragma solidity ^0.8.0;
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

By including all required compiler options and flags in a single pragma directive, the potential conflicts could be avoided and ensure that the contract can be compiled correctly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	LaunchPad.sol#L376,416
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 ^0.8.0;
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	LaunchPad.sol#L1631
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
_token.transfer(Data.withdrawer,
_token.balanceOf(address(this)))
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Address	Library			
	isContract	Internal		
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	verifyCallResultFromTarget	Internal		
	_revert	Private		
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	_callOptionalReturn	Private	✓	
	_callOptionalReturnBool	Private	✓	
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		

IERC20	Interface			
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
ERC20	Implementation	Context, IERC20		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-

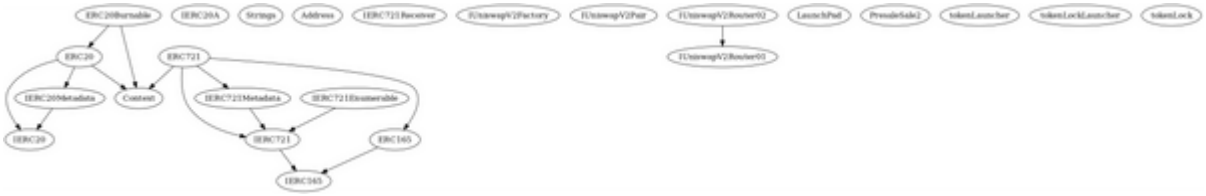
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
IUniswapV2Factory	Interface			
	getPair	External		-
IUniswapV2Pair	Interface			
	approve	External	✓	-
IUniswapV2Router02	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-

	addLiquidityETH	External	Payable	-
IERC721	Interface			
	balanceOf	External		-
	ownerOf	External		-
	getTokenIds	External		-
	getApproved	External		-
	safeTransferFrom	External	✓	-
LaunchPad	Implementation			
		Public	✓	-
	changeFeeForPoolCreation	Public	✓	onlyAdmin
	changeReceiver	Public	✓	onlyAdmin
	createPresale	Public	Payable	-
	getFee	Public		-
	getPoolDetails	Public		-
	getPaginatedPoolDetails	Public		-
	getTiers	Public		-
	changeTiers	Public	✓	onlyAdmin
PresaleContract	Implementation			

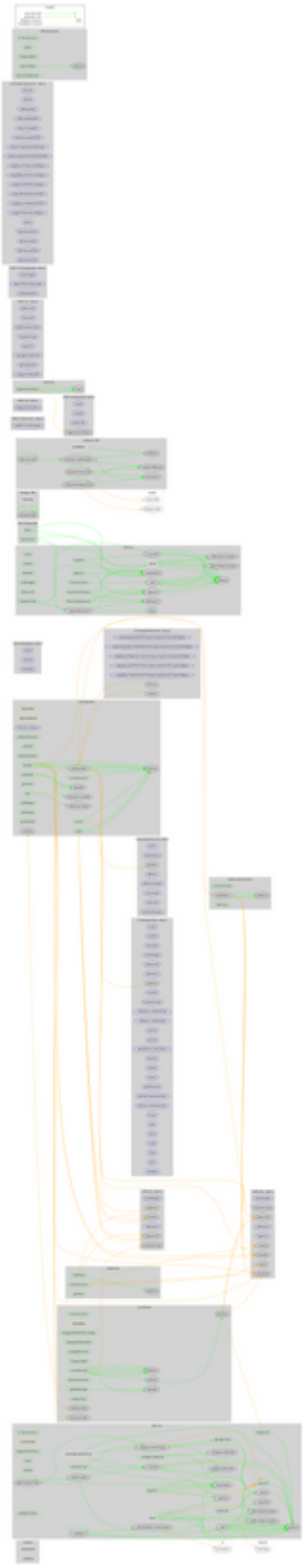
		Public	✓	-
	adminAllowance	Public	✓	onlyAdmin
	editPool	Public	✓	_adminAllowed
	claim	Public	✓	-
	getEntitlement	Public		-
	withdraw	Public	Payable	-
	buy	Public	Payable	_adminAllowed
	getContributors	Public		-
	finalize	Public	✓	-
	cancel	Public	✓	-
	addLiquidity	Public	✓	-
	addBadges	Public	✓	onlyAdmin
	getBadges	Public		-
	getSelfInfo	Public		-
	deemedCancelled	Public		-
		External	Payable	-
TokenLauncher	Implementation			
		Public	✓	-
	setfee	Public	✓	-
	changeAdmin	Public	✓	-
	launchToken	Public	Payable	-

	getUsertokenList	Public		-
	getAllTokens	Public		-
		External	Payable	-
TokenLockLauncher	Implementation			
		Public	✓	-
	lockToken	Public	✓	-
	getAllLocks	Public		-
tokenLock	Implementation			
		Public	✓	-
	withdraw	Public	✓	-
	getData	Public		-

Inheritance Graph



Flow Graph



Summary

Ratpad contract implements a token, locker, and utility 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