Cyberscope

Audit Report
UPAY

September 2024

Network	BSC
Address	0xe2ccc252b10bd17bde1f48a53260947abfc12862
Audited by	© cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Unresolved
●	BC	Blacklists Addresses	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	TSD	Total Supply Diversion	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	IAER	Inefficient Array Element Removal	Unresolved
●	MTEE	Missing Transfer Event Emission	Unresolved
●	MU	Modifiers Usage	Unresolved
●	ORBF	Owner Restricted Burn Function	Unresolved
●	UT	Unchecked Timelock	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	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	UPAY
Compiler Version	v0.8.26+commit.8a97fa7a
Optimization	200 runs
Explorer	https://bscscan.com/address/0xe2ccc252b10bd17bde1f48a53260947abfc12862
Address	0xe2ccc252b10bd17bde1f48a53260947abfc12862
Network	BSC
Symbol	UPAY
Decimals	18
Total Supply	2,000,000,000

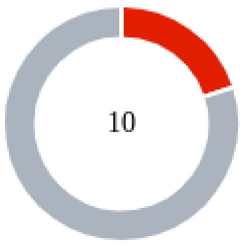
Audit Updates

Initial Audit	10 Sep 2024

Source Files

Filename	SHA256
UPAY.sol	6647f2bad01acf9b118e35e6f8d56acdd1a97465fa821d45429056fec49aafdc

Findings Breakdown



●	Critical	2
●	Medium	0
●	Minor / Informative	8

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

BT - Burns Tokens

Criticality	Critical
Location	UPAY.sol#L82
Status	Unresolved

Description

The contract owner has the authority to burn tokens from a specific address. The owner may take advantage of it by calling the `lockTokens` function with arguments an address and an arbitrary time duration. As a result, the targeted address will lose access to the corresponding tokens for the specified duration. The burned tokens can be restored to the targeted address by the owner using the `unlockTokens` function, only after the specified period has elapsed.

```
function lockTokens(address _account, uint256 _amount, uint256
_releaseTime) public onlyOwner {
        require(_amount <= balanceOf[_account], "Insufficient
balance");
        balanceOf[_account] = balanceOf[_account] - _amount;
        lockups[_account].push(Lockup(_amount, _releaseTime));
        emit Lock(msg.sender, _account, _amount, _releaseTime);
}
```

```
function unlockTokens(address _account, uint256 _index) public
onlyOwner {
        require(_index < lockups[_account].length, "Invalid
index");
        require(block.timestamp >=
lockups[_account][_index].releaseTime, "Tokens are still
locked");

        uint256 amount = lockups[_account][_index].amount;

        // Remove the lockup
        for (uint256 i = _index; i < lockups[_account].length -
1; i++) {
            lockups[_account][i] = lockups[_account][i + 1];
        }
        lockups[_account].pop();

        balanceOf[_account] = balanceOf[_account] + amount;

        emit Unlock(msg.sender, _account, amount);
    }
```

However, the contract owner has the authority to permanently burn tokens from a user's balance by passing an infinitely long `_releaseTime` to the `lockTokens` function. In this case, the specified amount can be indefinitely irrecoverable from the user's balance.

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 will address the threats but it is irreversible. To avoid loss of funds, the contract must not include locked tokens when ownership is renounced.

TSD - Total Supply Diversion

Criticality	Minor / Informative
Location	UPAY.sol#L82
Status	Unresolved

Description

The total supply of a token is the total number of tokens that have been created, while the balances of individual accounts represent the number of tokens that an account owns. The total supply and the balances of individual accounts are two separate concepts that are managed by different variables in a smart contract. These two entities should be equal to each other.

In the contract, the amount that is subtracted from the balance of an account is not subtracted from the total supply. As a result, the sum of balances is diverse from the total supply.

```
function lockTokens(address _account, uint256 _amount, uint256
_releaseTime) public onlyOwner {
    ...
    balanceOf[_account] = balanceOf[_account] - _amount;
    lockups[_account].push(Lockup(_amount, _releaseTime));
    ...
}
```

Recommendation

The total supply and the balance variables are separate and independent from each other. The total supply represents the total number of tokens that have been created, while the balance mapping stores the number of tokens that each account owns. The sum of balances should always equal the total supply.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	UPAY.sol#L39
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	UPAY.sol#L98
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 up 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.

```
function unlockTokens(address _account, uint256 _index) public
onlyOwner {

        ...
        for (uint256 i = _index; i < lockups[_account].length -
1; i++) {
            lockups[_account][i] = lockups[_account][i + 1];
        }
        lockups[_account].pop();
        ...
}
```

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.

MTEE - Missing Transfer Event Emission

Criticality	Minor / Informative
Location	UPAY.sol#L40,76,85,103
Status	Unresolved

Description

The contract does not emit an event when portions of the main amount are transferred during the transfer process. This lack of event emission results in decreased transparency and traceability regarding the flow of tokens, and hinders the ability of decentralized applications (dApps), such as blockchain explorers, to accurately track and analyze these transactions. According to the ERC20 standard, transfers of value must fire the Transfer event. This discrepancy can lead to inconsistencies and incompatibilities with other contracts.

```
balanceOf[msg.sender] = totalSupply;
```

```
balanceOf[_account] = balanceOf[_account] + amount;
```

```
balanceOf[msg.sender] = balanceOf[msg.sender] - _value;
```

Recommendation

It is advisable to incorporate the emission of detailed event logs following each asset transfer. These logs should encapsulate key transaction details, including the identities of the sender and receiver, and the quantity of assets transferred. Implementing this practice will enhance the reliability and transparency of transaction tracking systems, ensuring accurate data availability for ecosystem participants. The team is therefore advised to review and revise the implementation of the transfer mechanism to ensure full compliance with the ERC20 standard. https://eips.ethereum.org/EIPS/eip-20.

MU - Modifiers Usage

Criticality	Minor / Informative
Location	UPAY.sol#L92,109
Status	Unresolved

Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

```
require(_index < lockups[_account].length, "Invalid index");
require(block.timestamp >= lockups[_account][_index].releaseTime,
"Tokens are still locked");
```

Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

ORBF - Owner Restricted Burn Function

Criticality	Minor / Informative
Location	UPAY.sol#L73
Status	Unresolved

Description

The contract implements a burn function that enables the owner to burn tokens from their own balance. However, access to this function is restricted to the contract owner only, preventing any other addresses from using it. Typically individual users should have the ability to burn tokens from their own balances.

```
function burn(uint256 _value) public onlyOwner {
    require(_value <= balanceOf[msg.sender], "Insufficient balance");
    balanceOf[msg.sender] = balanceOf[msg.sender] - _value;
    totalSupply = totalSupply - _value;
    emit Burn(msg.sender, _value);
}
```

Recommendation

The team is advised to review the burn mechanism to confirm that its current behavior aligns with the intended design.

UT - Unchecked Timelock

Criticality	Minor / Informative
Location	UPAY.sol#L82
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. Specifically the `_releaseTime` passed as an argument in the `lockTokens` functions is not checked to ensure that it corresponds to a future time instance and the difference from the current timestamp does not exceed some acceptable limits.

```
function lockTokens(address _account, uint256 _amount, uint256
_releaseTime) public onlyOwner {
        ...
    }
```

Recommendation

The team is advised to properly check the variables according to the required specifications. In particular, the contract should incorporate a conditional check on the `_releaseTime` to ensure that the provided timestamp falls within acceptable limits.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	UPAY.sol#L9,10,11
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.

```
string public name = "UPAY"
string public symbol = "UPAY"
uint8 public decimals = 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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	UPAY.sol#L43,58,64,73,82,91,108,119
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.

```
uint256 _value
address _to
address _spender
address _from
address _account
uint256 _releaseTime
uint256 _amount
uint256 _index
```

Recommendation

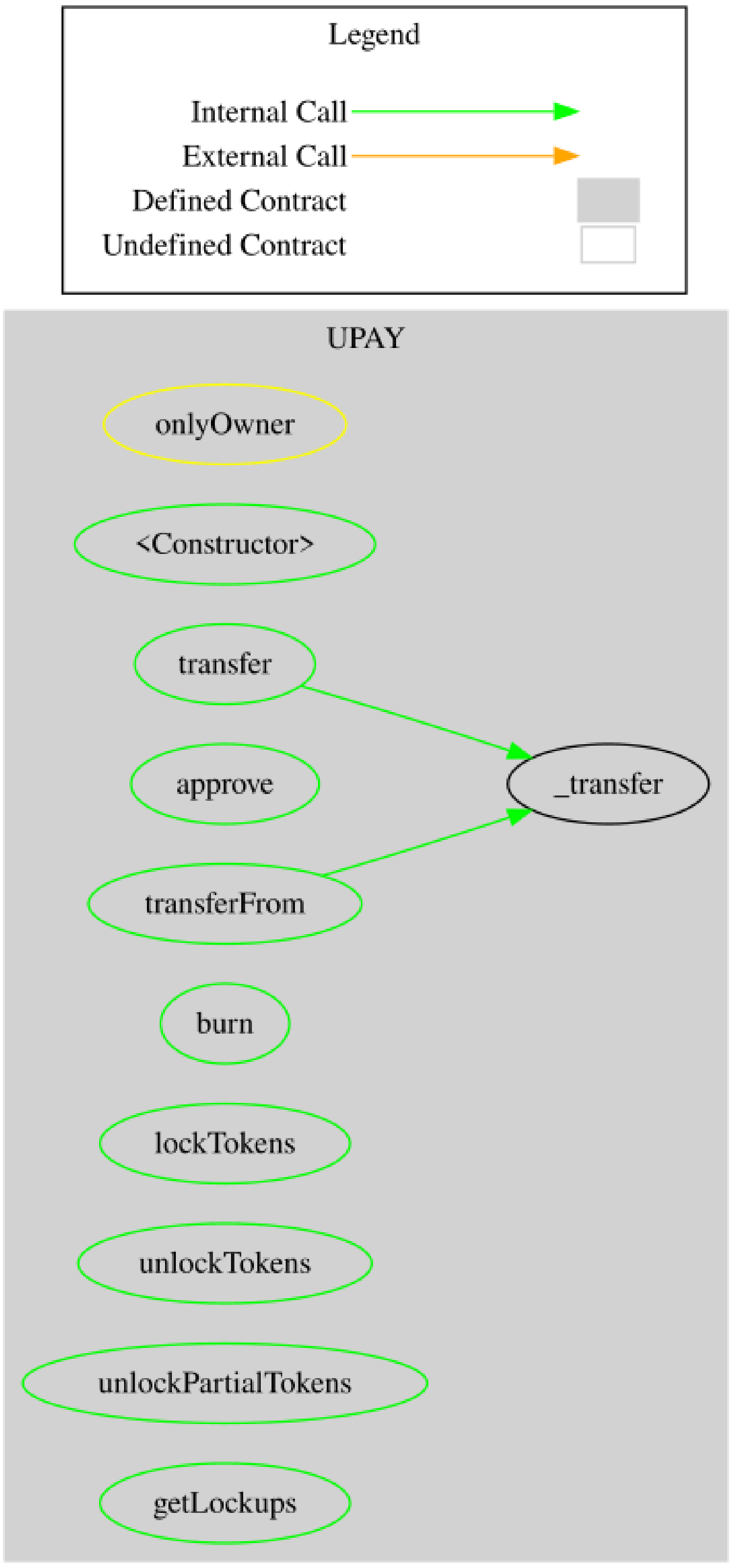
By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/stable/style-guide.html#naming-conventions.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
UPAY	Implementation			
		Public	✓	-
	transfer	Public	✓	-
	_transfer	Internal	✓	
	approve	Public	✓	-
	transferFrom	Public	✓	-
	burn	Public	✓	onlyOwner
	lockTokens	Public	✓	onlyOwner
	unlockTokens	Public	✓	onlyOwner
	unlockPartialTokens	Public	✓	onlyOwner
	getLockups	Public		onlyOwner

Inheritance Graph

Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
UPAY
onlyOwner
<Constructor>
transfer
approve
transferFrom
burn
lockTokens
unlockTokens
unlockPartialTokens
getLockups
_transfer

Summary

UPAY contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner to burn and lock tokens from any address. if the contract owner abuses this functionality, then the users could lose their tokens. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

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