Cyberscope

Audit Report

HyDRAULIC

June 2024

Repository https://github.com/SololPmanagement/hydraulic-contracts-audit-internal

Commit 73d9cad07088aed0be8fe0b014ab9a63932283d8

Audited by © cyberscope

Table of Contents

Review

Contract Name	DRAU
Repository	https://github.com/SoloIPmanagement/hydraulic-contracts-audit-internal
Commit	73d9cad07088aed0be8fe0b014ab9a63932283d8
Testing Deploy	https://testnet.bscscan.com/address/0xf8c0c51ff4436a7e8edd4c00a555cada865b63a1
Decimals	18

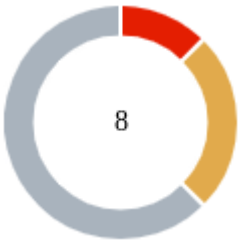
Audit Updates

Initial Audit	13 Jun 2024

Source Files

Filename	SHA256
contracts/DRAU.sol	f4719747ef6c472848d4d0f9e79714c00c34f83e11b9f5b38c3ffc7163eef2be

Findings Breakdown



Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	2	0	0	0
Minor / Informative	5	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	OTUT	Transfers User's Tokens	Unresolved
●	BC	Blacklists Addresses	Unresolved
●	ST	Stops Transactions	Unresolved
●	IBL	Inconsistent Burn Logic	Unresolved
●	MT	Mints Tokens	Unresolved
●	MU	Modifiers Usage	Unresolved
●	RNRM	Redundant No Reentrant Modifier	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved

OTUT - Transfers User's Tokens

Criticality	Critical
Location	contracts/DRAU.sol#L60,75
Status	Unresolved

Description

The contract owner has the ability to remove tokens from any address. This can be executed through the `wipeBlacklistedAddress` or `burn` function. As a result, the owner may potentially take advantage this capability, leading to users losing their tokens without their consent or any notification.

```
   function wipeBlacklistedAddress(address _address) external
onlyOwner {
        if(!blacklisted[_address]) {
            revert IsNotBlacklistedAddress(_address);
        }
        _burn(_address, balanceOf(_address));
        emit BlacklistedAddressWiped(_address);
    }

    function burn(address from, uint256 amount) external
onlyOwner nonReentrant {
        _burn(from, amount);
    }
```

Recommendation

It is recommended to implement stricter access controls and oversight mechanisms to ensure that these functions are only used in legitimate and transparent circumstances. The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

BC - Blacklists Addresses

Criticality	Medium
Location	contracts/DRAU.sol#L44
Status	Unresolved

Description

The contract owner has the authority to stop addresses from transactions. The owner may take advantage of it by calling the `blacklist` function.

```
   function blacklist(address _address) external
isNotBlacklisted(_address) onlyOwner {
        if (_address == address(0)) {
            revert IsInvalidAddress(_address);
        }
        blacklisted[_address] = true;
        emit AddressBlacklisted(_address);
    }
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

ST - Stops Transactions

Criticality	Medium
Location	contracts/DRAU.sol#L79,83
Status	Unresolved

Description

The contract owner has the authority to stop the transactions for all users excluding the owner. The owner may take advantage calling the `pause` method. As a result, the transactions of all the users will fail, disrupting normal usage and potentially causing significant inconvenience or financial loss.

```
   function transfer(address to, uint256 amount) public
override onlyOwnerWhenPaused isNotBlacklisted(msg.sender)
isNotBlacklisted(to) nonReentrant returns (bool) {
        return super.transfer(to, amount);
    }

    function transferFrom(address from, address to, uint256
amount) public override onlyOwnerWhenPaused
isNotBlacklisted(msg.sender) isNotBlacklisted(from)
isNotBlacklisted(to) nonReentrant returns (bool) {
        return super.transferFrom(from, to, amount);
    }

    function pause() external onlyOwner {
        _pause();
    }
```

Recommendation

It is recommended to implement stricter controls and oversight mechanisms around the `pause` function to prevent potential misuse. Additionally, the team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

IBL - Inconsistent Burn Logic

Criticality	Minor / Informative
Location	contracts/DRAU.sol#L60,75
Status	Unresolved

Description

The contract is designed with a `burn` function that gives the contract owner the authority to burn tokens from any address. Consequently, the `wipeBlacklistedAddress` function, which allows the burning of tokens only from blacklisted addresses, becomes redundant since any address can be targeted by the `burn` function.

```
   function wipeBlacklistedAddress(address _address) external
onlyOwner {
        if(!blacklisted[_address]) {
            revert IsNotBlacklistedAddress(_address);
        }
        _burn(_address, balanceOf(_address));
        emit BlacklistedAddressWiped(_address);
    }

    function burn(address from, uint256 amount) external
onlyOwner nonReentrant {
        _burn(from, amount);
    }
```

Recommendation

It is recommended to reconsider the actual logic and intended scenarios for these functions. If the intent is to allow burning tokens only from blacklisted addresses, then the `burn` function should be restricted accordingly. Otherwise, if broader `burn` capabilities are desired, the redundancy with the `wipeBlacklistedAddress` function should be addressed to maintain clear and coherent contract logic.

MT - Mints Tokens

Criticality	Minor / Informative
Location	contracts/DRAU.sol#L68
Status	Unresolved

Description

The contract owner has the authority to mint tokens until the `MAXIMUM_SUPPLY` is reached. The owner may take advantage of it by calling the `mint` function. As a result, the contract tokens will be highly inflated.

```
    function mint(address to, uint256 amount) external onlyOwner
nonReentrant {
        if (totalSupply() + amount > MAXIMUM_SUPPLY) {
            revert MaximumSupplyExceeded();
        }
        _mint(to, amount);
    }
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.
- Minth the `MAXIMUM_SUPPLY` amount of tokens.

MU - Modifiers Usage

Criticality	Minor / Informative
Location	contracts/DRAU.sol#L53,61
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.

```
if(!blacklisted[_address]) {
    revert IsNotBlacklistedAddress(_address);
}
```

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.

RNRM - Redundant No Reentrant Modifier

Criticality	Minor / Informative
Location	contracts/DRAU.sol#L68,75,79,83
Status	Unresolved

Description

The contract uses the `nonReentrant` modifier to the `mint` , `burn` , `transfer` and `transferFrom` functions, which suggests an intention to prevent potential reentrancy attacks. However, neither of these functions deals with the transfer of the native token or any other value. As such, the risk of reentrancy attacks in these specific functions is minimal to non-existent.

```
    function mint(address to, uint256 amount) external onlyOwner
nonReentrant {
        if (totalSupply() + amount > MAXIMUM_SUPPLY) {
            revert MaximumSupplyExceeded();
        }
        _mint(to, amount);
    }

    function burn(address from, uint256 amount) external
onlyOwner nonReentrant {
        _burn(from, amount);
    }

    function transfer(address to, uint256 amount) public
override onlyOwnerWhenPaused isNotBlacklisted(msg.sender)
isNotBlacklisted(to) nonReentrant returns (bool) {
        return super.transfer(to, amount);
    }

    function transferFrom(address from, address to, uint256
amount) public override onlyOwnerWhenPaused
isNotBlacklisted(msg.sender) isNotBlacklisted(from)
isNotBlacklisted(to) nonReentrant returns (bool) {
        return super.transferFrom(from, to, amount);
    }
```

Recommendation

To address this finding and enhance code simplicity and clarity, it is recommended to remove the unnecessary `nonReentrant` modifier from the above mentioned functions. By removing the modifier, the code becomes more streamlined and easier to comprehend, reducing the gas consuption.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/DRAU.sol#L14,44,52,60
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 public MAXIMUM_SUPPLY
address _address
```

Recommendation

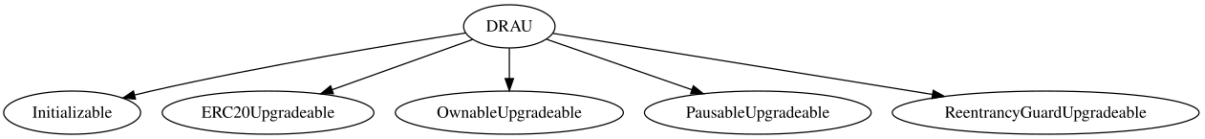
By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

Functions Analysis

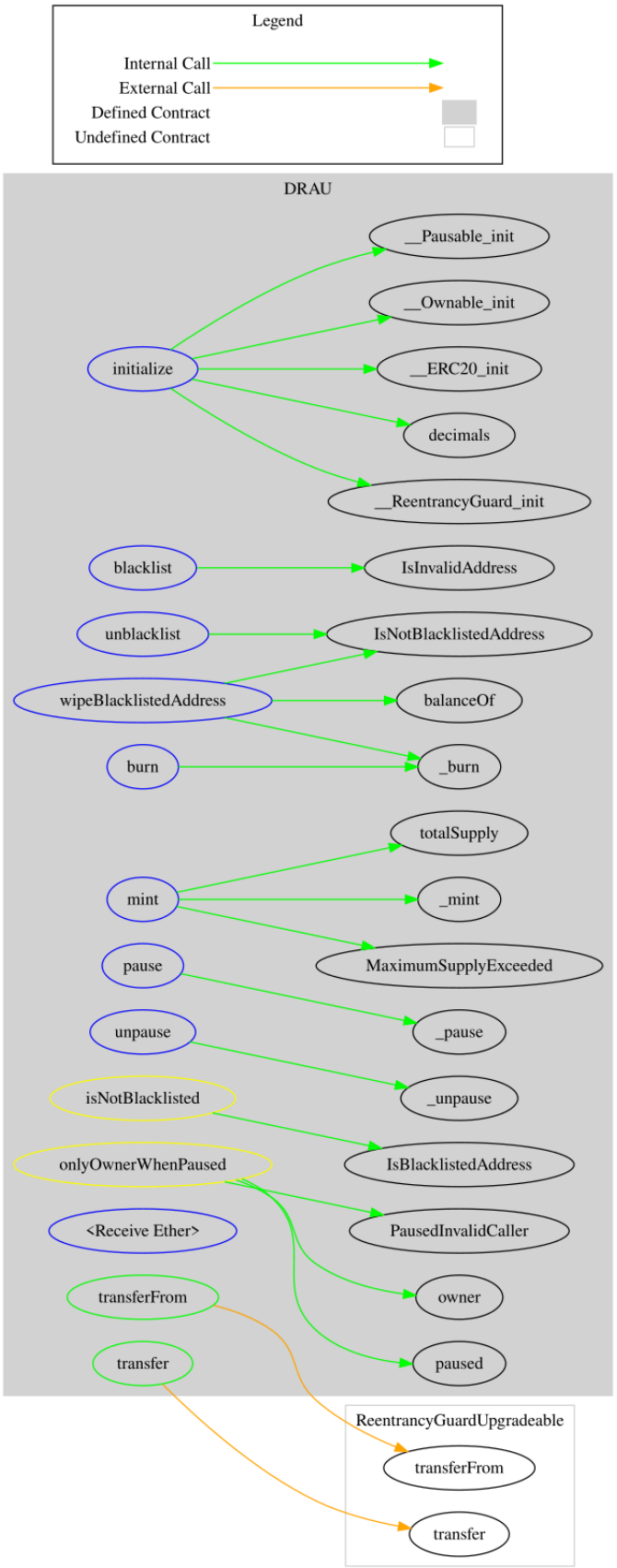
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
DRAU	Implementation	Initializable, ERC20Upgradeable, OwnableUpgradeable, PausableUpgradeable, ReentrancyGuardUpgradeable		
	initialize	External	✓	initializer
	blacklist	External	✓	isNotBlacklisted onlyOwner
	unblacklist	External	✓	onlyOwner
	wipeBlacklistedAddress	External	✓	onlyOwner
	mint	External	✓	onlyOwner nonReentrant
	burn	External	✓	onlyOwner nonReentrant
	transfer	Public	✓	onlyOwnerWhenPaused isNotBlacklisted isNotBlacklisted nonReentrant
	transferFrom	Public	✓	onlyOwnerWhenPaused isNotBlacklisted isNotBlacklisted isNotBlacklisted nonReentrant
	pause	External	✓	onlyOwner
	unpause	External	✓	onlyOwner

		External	Payable	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
DRAU
__Pausable_init
__Ownable_init
initialize
__ERC20_init
decimals
__ReentrancyGuard_init
blacklist
IsInvalidAddress
unblacklist
IsNotBlacklistedAddress
wipeBlacklistedAddress
balanceOf
burn
_burn
totalSupply
mint
_mint
pause
MaximumSupplyExceeded
_pause
unpause
_unpause
isNotBlacklisted
onlyOwnerWhenPaused
IsBlacklistedAddress
<Receive Ether>
PausedInvalidCaller
transferFrom
owner
transfer
paused
ReentrancyGuardUpgradeable
transferFrom
transfer

Summary

HyDRAULIC 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 like stop transactions, burn the user's tokens, mint tokens, and blacklist addresses. 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

https://www.cyberscope.io