Cyberscope

Audit Report

HYDT Stablecoin

July 2024

Network	BSC
Address	0x6e98C28c210f74Adf54b20138A64EE942A8DBD76
Audited by	© cyberscope

Table of Contents

Review

Explorer	https://bscscan.com/address/0x6e98c28c210f74adf54b20138a64ee942a8dbd76

Audit Updates

Initial Audit	26 Jun 2024 https://github.com/cyberscope-io/audits/blob/main/hydt/v1/audit.pdf
Corrected Phase 2	29 Jul 2024

Source Files

Filename	SHA256
contracts/AffiliateWithdrawal.sol	cecc50fb8acbbaf5942975c7cf3c132ad5eea5b4fcfa22a8dacef9ca3c227082

Overview

The `AffiliateWithdrawal` smart contract is designed to manage and facilitate the withdrawal of tokens under specific conditions, including immediate withdrawals and those subject to vesting periods. This contract provides a structured way for users to claim and manage their token rewards, using the HYDT and HYGT tokens. The contract also incorporates security features like signature verification to ensure that withdrawals are authorized and conform to predefined terms.

claimWithdraw Functionality

The `claimWithdraw` function is central to the contract, allowing users to initiate the withdrawal process based on specific parameters. This function supports two main withdrawal types, immediate and vested. For immediate withdrawals, tokens are released instantly without any multipliers. However, if a vesting option is selected (e.g. 3 months or 12 months), the contract only immediately releases the HYDT tokens and records the withdrawal details, including the user's information and vesting period, into a mapping. The HYGT tokens under vesting are locked and can only be claimed after the specified period has elapsed, leveraging different multipliers based on the duration of the vesting.

withdraw Functionality

Once the vesting period is complete, users can claim their vested tokens through the `withdraw` function. This function is essential for users who selected a vesting option during their initial withdrawal request. It calculates the final amount of HYGT tokens to be released by applying the appropriate multiplier corresponding to the selected vesting period. The function ensures that the withdrawal conditions are met, including the passage of the vesting period, and then proceeds to mint and release the vested HYGT tokens to the user's address.

Owner Functionality

The owner of the contract has the authority to update the `signerAddress`, which plays a crusial role in verifying the data necessary for the operational integrity of the contract.

When a new `signerAddress` is set, the contract emits a `SignerAddressUpdated` event, which provides transparency and traceability of this significant action.

Roles

Owner

The owner can interact with the following functions:

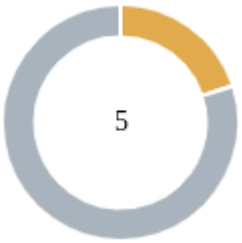
- function setSignerAddress

Users

The users can interact with the following functions:

- function claimWithdraw
- function withdraw

Findings Breakdown



Critical	0
Medium	1
Minor / Informative	4

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	1	0	0	0
Minor / Informative	4	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Medium	MLWB	Mint Limit Withdrawal Block	Unresolved
Minor / Informative	RVC	Redundant Vesting Check	Unresolved
Minor / Informative	IDI	Immutable Declaration Improvement	Unresolved
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor / Informative	L16	Validate Variable Setters	Unresolved

MLWB - Mint Limit Withdrawal Block

Criticality	Medium
Location	contracts/AffiliateWithdrawal.sol#L203
Status	Unresolved

Description

The contract is designed to mint `HYGT` tokens during the immediate withdrawal of funds, but it does not account for the maximum token supply cap of the `HYGT` token. This oversight means that when the `HYGT` token's predefined maximum supply limit is reached, any further attempts to mint tokens will fail. Consequently, this failure will also prevent the completion of the withdrawal process, as the minting step is integral to the withdrawal functionality. The absence of a check or handling mechanism for this condition can disrupt normal contract operations and adversely affect user transactions.

```
if (HYGTAmount > 0) {
        HYGT.mint(msg.sender, HYGTAmount);
}
```

Recommendation

It is recommended to consider and handle scenarios where the maximum token supply limit is reached. The contract could verify that the total supply of `HYGT` tokens has not exceeded the maximum limit before attempting to mint new tokens as part of the withdrawal process. Implementing this verification will prevent the mint function from failing and ensure that withdrawals can proceed without interruption. Additionally, appropriate error handling or alternative solutions should be designed to manage cases where the minting cannot be performed due to the max supply cap being reached.

RVC - Redundant Vesting Check

Criticality	Minor / Informative
Location	contracts/AffiliateWithdrawal.sol#L297,163,232
Status	Unresolved

Description

The contract is designed to handle token withdrawals with a focus on vesting periods through its `processVestingInfo` function. This function enforces valid vesting options (`vestingOption`) by verifying that the selected vesting duration matches predefined periods (firstVestingPeriod or secondVestingPeriod) before storing this information in the `userWithdrawals` mapping. This validation ensures that any stored vesting duration is already confirmed as valid. However, the subsequent `withdrawWithVesting` function redundantly checks these durations again even though they have been validated and stored previously. This redundancy in validation leads to unnecessary processing and gas expenditure when the `withdraw` function is called, as the vesting duration retrieved from the mapping will always be valid, having been checked at the point of initial storage.

```
    function processVestingInfo(
        ...
        uint256 vestingOption,
        ...
    ) private {
        uint256 _id = userTotalWithdrawals[msg.sender]++;

        if (
            vestingOption != firstVestingPeriod &&
            vestingOption != secondVestingPeriod
        ) {
            revert InvalidVestingDuration();
        }

        userWithdrawals[user][_id] = UserWithdrawal({
            ...
            vestingMonths: vestingOption,
            ...
        });
        ...

    function withdraw(uint256 _id) external {
        ...
        // Vesting period is over, claim funds
        withdrawWithVesting(
            _id,
            userWithdrawals[msg.sender][_id].HYGTAmount,
            userWithdrawals[msg.sender][_id].vestingMonths
        );
    }

    function withdrawWithVesting(
        uint256 _id,
        uint256 HYGTAmount,
        uint256 vestingDuration
    ) private {
        if (
            vestingDuration != firstVestingPeriod &&
            vestingDuration != secondVestingPeriod
        ) {
            revert InvalidVestingDuration();
        }
```

Recommendation

It is recommended to remove the conditional check in the `withdrawWithVesting` function that reverts if the vesting duration does not match the `firstVestingPeriod` or `secondVestingPeriod`. This check is unnecessary since the validation is already

handled in the `processVestingInfo` function. Removing this redundant condition will simplify the code, reduce gas costs, and eliminate an unnecessary validation step, thereby streamlining the contract's execution.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/AffiliateWithdrawal.sol#L67,68,69,70
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.

```
firstVestingPeriod
secondVestingPeriod
firstVestingmultiplier
secondVestingmultiplier
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/AffiliateWithdrawal.sol#L13,14,48,81,98,99,163,194,195,227,228,259,260,287,288
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.

```
IHYDT public HYDT
IHYGT public HYGT
event withdraw_Immediately(address, uint256, uint256);
address _addr
uint256 HYDTAmount
uint256 HYGTAmount
uint256 _id
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/AffiliateWithdrawal.sol#L66
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.

```
signerAddress = _signer
```

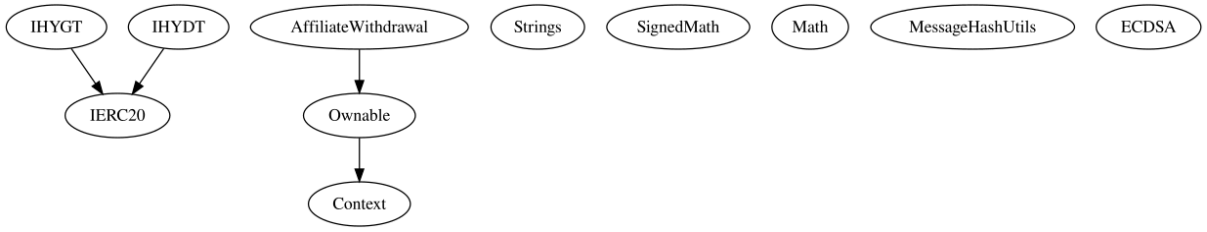
Recommendation

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.

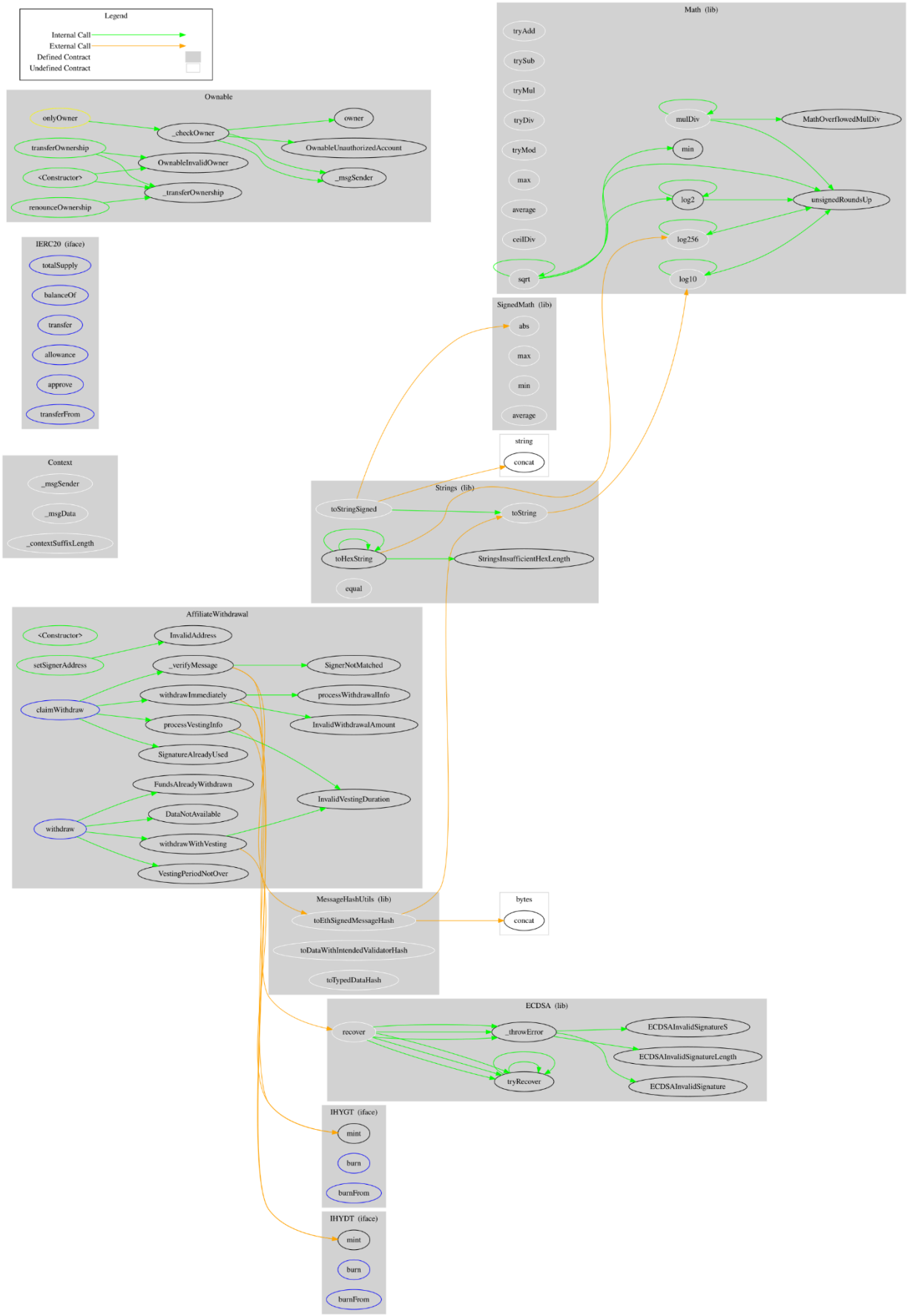
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
AffiliateWithdrawal	Implementation	Ownable		
		Public	✓	Ownable
	setSignerAddress	Public	✓	onlyOwner
	claimWithdraw	External	✓	-
	_verifyMessage	Private		
	withdraw	External	✓	-
	withdrawImmediately	Private	✓	
	withdrawWithVesting	Private	✓	
	processWithdrawalInfo	Private	✓	
	processVestingInfo	Private	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Ownable
onlyOwner
_checkOwner
owner
transferOwnership
OwnableInvalidOwner
OwnableUnauthorizedAccount
<Constructor>
_msgSender
renounceOwnership
_transferOwnership
Math (lib)
tryAdd
trySub
tryMul
tryDiv
tryMod
max
average
ceilDiv
mulDiv
MathOverflowedMulDiv
min
log2
unsignedRoundsUp
log256
log10
sqrt
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
SignedMath (lib)
abs
max
min
average
string
concat
Context
_msgSender
_msgData
_contextSuffixLength
Strings (lib)
toStringSigned
toString
toHexString
StringsInsufficientHexLength
equal
AffiliateWithdrawal
<Constructor>
InvalidAddress
setSignerAddress
_verifyMessage
SignerNotMatched
withdrawImmediately
processWithdrawalInfo
claimWithdraw
InvalidWithdrawalAmount
processVestingInfo
SignatureAlreadyUsed
FundsAlreadyWithdrawn
InvalidVestingDuration
DataNotAvailable
withdraw
withdrawWithVesting
VestingPeriodNotOver
MessageHashUtils (lib)
bytes
toEthSignedMessageHash
concat
toDataWithIntendedValidatorHash
toTypedDataHash
ECDSA (lib)
recover
_throwError
ECDSAInvalidSignatureS
ECDSAInvalidSignatureLength
tryRecover
ECDSAInvalidSignature
IHYGT (iface)
mint
burn
burnFrom
IHYDT (iface)
mint
burn
burnFrom

Summary

The AffiliateWithdrawal contract implements a withdrawal mechanism for the HYDT and HYGT tokens, facilitating user withdrawals with options for immediate or vested disbursements. This audit investigates security issues, business logic concerns, and potential improvements to ensure the contract's functionality and user protection.

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