AudiTBlock

HELLDIVER_Presale

v0.8.20+commit.7dd6d404

v0.8.20

Low-Risk	Medium-Risk	High-Risk
Low-risk code	Medium-risk code	High-risk code

bsc-testnet

0x9d918588c8fd6d7f294125f3bd1534c641b888df

[Disclaimer]

Types of Severities

High

A high-severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low

Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Techniques and Methods

The overall quality of code.

- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrance and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, and their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

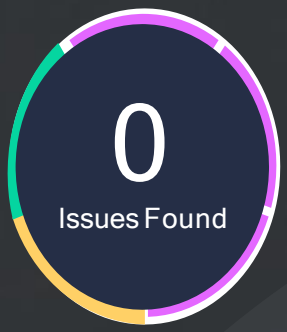
Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms Used for Audit

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Name	HELLDIVER_Presale
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyze the contract codebase for quality, security, and correctness.
Audit Team	AuditBlock



High Medium Low Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	2
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	0	0	0

ID	File Name	Audit Status
10022	HELLDIVER_Presale.sol	Pass

Smart Contract Weakness Classification (SWC) Vulnerabilities for Attacks

- ✗ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ Exception Disorder
- ✓ Gasless Send
- ✓ Use of tx.origin
- ✓ Compiler version not fixed
- ✓ Address hardcoded
- ✗ Divide before multiply
- ✓ Integer overflow/underflow
- ✓ Dangerous strict equalities
- ✓ Tautology or contradiction
- ✓ Missing Zero Address Validation
- ✓ Return values of low-level calls
- ✓ Revert/require functions
- ✓ Private modifier
- ✓ Using block.timestamp
- ✓ Multiple Sends
- ✓ Using SHA3
- ✓ Using suicide
- ✓ Using throw
- ✓ Using inline assembly

Phase 1

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Severity Issues

1. divide-before-multiply (Medium)

```
710 ~    {
711          uint256 bnbToUsdt = (_amount * (getLatestPrice())) / (1 ether);
712          uint256 numberOfTokens = (bnbToUsdt * price) / (1e8);
713          return numberOfTokens;
714      }
```

Description

Our auditor identified that your contract exhibits Solidity's integer truncation behavior. This means that performing division before multiplication can lead to precision loss.

Recommendation

It is important to double-check your contract. Consider reordering operations to prioritize multiplication before division.

Status

Acknowledged

2. Reentrancy vulnerabilities (Medium)

```
675             (bool success) = mainToken.transfer(msg.sender, _reward);
676             require(success, "token transfer failed");
677             userStake.claimedReward += _reward;
678         }
679         (bool status) = mainToken.transfer(msg.sender, userStake.stakedTokens);
```

Description

During their audit, our auditors identified a reentrancy bug in your contract. Note that reentrancies involving Ether are not reported (see reentrancy-eth for details).

Recommendation

It is important to double-check your contract and apply the check-effects-interactions pattern.

Status

Acknowledged

https://auditblock.report

Phase 2

HELLDIVER_Presale.nativeToToken(uint256,uint256) (contracts/HELLDIVER_Presale.sol#706-714) performs a multiplication on the result of a division:
- bnbToUsdt = (_amount * (getLatestPrice())) / (1000000000000000000) (contracts/HELLDIVER_Presale.sol#711)
- numberOfTokens = (bnbToUsdt * price) / (1e8) (contracts/HELLDIVER_Presale.sol#712)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

HELLDIVER_Presale.userStakes (contracts/HELLDIVER_Presale.sol#505) can be used in cross function reentrancies:
- HELLDIVER_Presale.calculateReward(address,uint256) (contracts/HELLDIVER_Presale.sol#657-667)
- HELLDIVER_Presale.userStakes (contracts/HELLDIVER_Presale.sol#505)
- userStake.unstakeTime = block.timestamp (contracts/HELLDIVER_Presale.sol#682)

HELLDIVER_Presale.userStakes (contracts/HELLDIVER_Presale.sol#505) can be used in cross function reentrancies:
- HELLDIVER_Presale.calculateReward(address,uint256) (contracts/HELLDIVER_Presale.sol#657-667)
- HELLDIVER_Presale.userStakes (contracts/HELLDIVER_Presale.sol#505)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

HELLDIVER_Presale.updateInfos(uint256,uint256,uint256,uint256,uint256) (contracts/HELLDIVER_Presale.sol#732-744) should emit an event for:
- soldToken = _sold (contracts/HELLDIVER_Presale.sol#739)
- amountRaised = _raised (contracts/HELLDIVER_Presale.sol#740)
- amountRaisedUSDT = _raisedInUsdt (contracts/HELLDIVER_Presale.sol#741)
- amountRaisedUSDC = _raisedInUsdc (contracts/HELLDIVER_Presale.sol#742)
- amountRaisedOverall = _amountRaisedOverall (contracts/HELLDIVER_Presale.sol#743)

HELLDIVER_Presale.changeAPY(uint256) (contracts/HELLDIVER_Presale.sol#759-761) should emit an event for:
- APY = _APY (contracts/HELLDIVER_Presale.sol#760)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

Ownable2Step.transferOwnership(address).newOwner (contracts/HELLDIVER_Presale.sol#166) lacks a zero-check on :
- _pendingOwner = newOwner (contracts/HELLDIVER_Presale.sol#167)

HELLDIVER_Presale.constructor(IERC20,address,address,address,address)._fundReceiver (contracts/HELLDIVER_Presale.sol#514) lacks a zero-check on :
- fundReceiver = address(_fundReceiver) (contracts/HELLDIVER_Presale.sol#518)

HELLDIVER_Presale.changeFundReceiver(address)._addr (contracts/HELLDIVER_Presale.sol#770) lacks a zero-check on :
- fundReceiver = address(_addr) (contracts/HELLDIVER_Presale.sol#771)

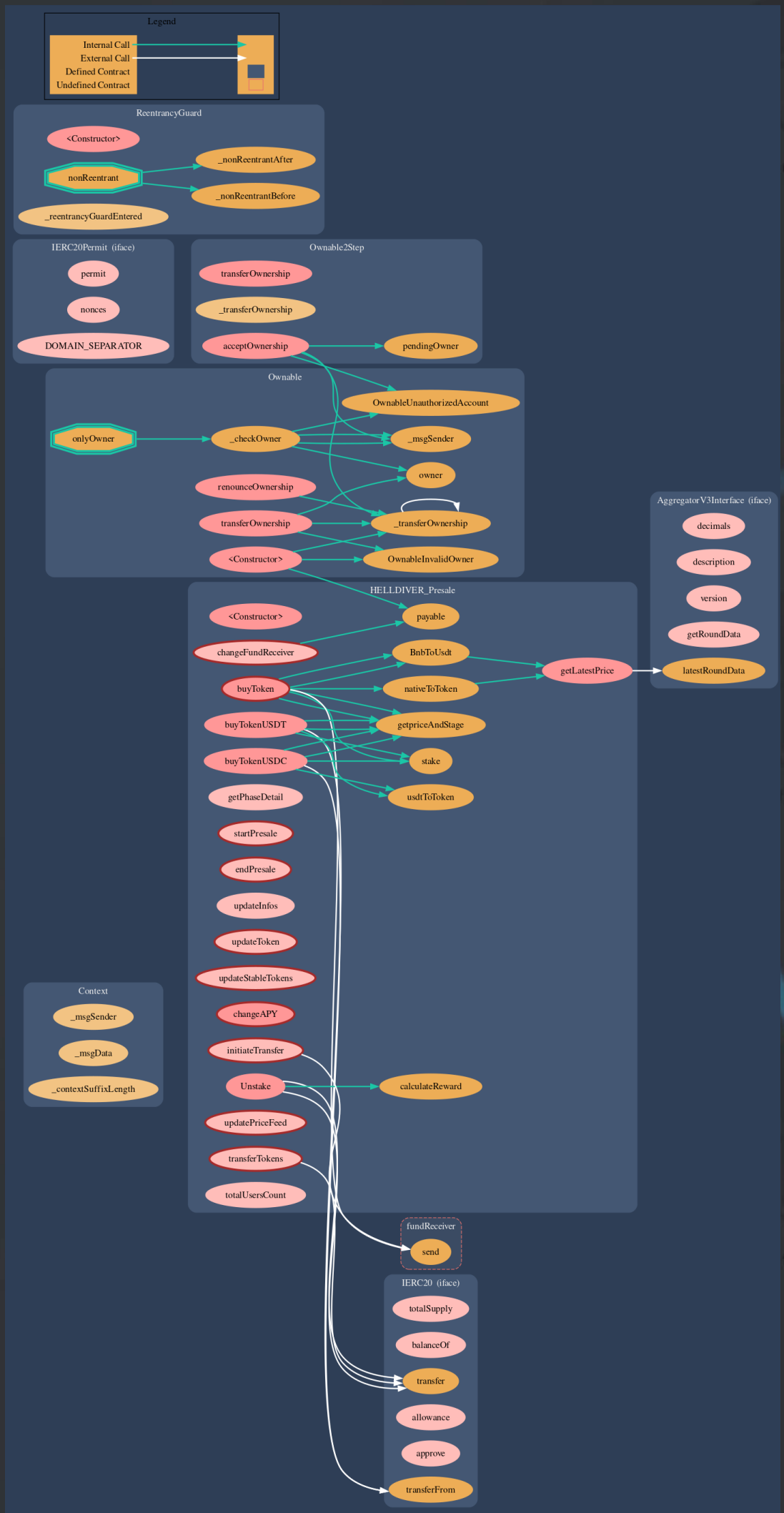
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

HELLDIVER_Presale.percentDivider (contracts/HELLDIVER_Presale.sol#480) should be constant
HELLDIVER_Presale.timeStep (contracts/HELLDIVER_Presale.sol#481) should be constant
HELLDIVER_Presale.tokensToSell (contracts/HELLDIVER_Presale.sol#482) should be constant
HELLDIVER_Presale.totalStages (contracts/HELLDIVER_Presale.sol#478) should be constant

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant

HELLDIVER_Presale.slitherConstructorVariables() (contracts/HELLDIVER_Presale.sol#434-793) uses literals with too many digits:
- prices = (77579519006982,52328623757195,39463299131807,31685678073510,26469031233456,22727272727272,19912385503783,17717930545712,15956598053295,14515894904920,13313806417254,12295585884667,11420740063956,10663254425250,10000000000000) (contracts/HELLDIVER_Presale.sol#461-477)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
ReentrancyGuard
<Constructor>
nonReentrant
_nonReentrantAfter
_nonReentrantBefore
_reentrancyGuardEntered
IERC20Permit (iface)
permit
nonces
DOMAIN_SEPARATOR
Ownable2Step
transferOwnership
_transferOwnership
acceptOwnership
pendingOwner
Ownable
OwnableUnauthorizedAccount
onlyOwner
_checkOwner
_msgSender
owner
renounceOwnership
transferOwnership
_transferOwnership
<Constructor>
OwnableInvalidOwner
AggregatorV3Interface (iface)
decimals
description
version
getRoundData
latestRoundData
HELLDIVER_Presale
<Constructor>
payable
changeFundReceiver
BnbToUsdt
getLatestPrice
buyToken
nativeToToken
buyTokenUSDT
getpriceAndStage
buyTokenUSDC
stake
getPhaseDetail
usdtToToken
startPresale
endPresale
updateInfos
updateToken
updateStableTokens
changeAPY
initiateTransfer
Unstake
calculateReward
updatePriceFeed
transferTokens
totalUsersCount
Context
_msgSender
_msgData
_contextSuffixLength
fundReceiver
send
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom

Phase 3

Closing Summary

In this report, we have considered the security of this **HELLDIVER_Presale**
We performed our audit according to the procedure described above.

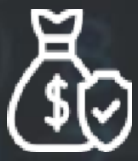
Several issues were identified during the audit and classified by severity. Recommendations and best practices were provided to improve code quality and security posture. The team has acknowledged all findings.

Disclaimer

AuditBlock

AuditBlock is a blockchain security company that provides professional services and solutions for securing blockchain projects. They specialize in smart contract audits on various blockchains and offer a range of services

100+
Audits Completed



$1M
Secured

100K
Lines of Code Audited

https://auditblock.report/

https://t.me/Auditblock

https://github.com/AuditBlock

https://twitter.com/oAuditBlock