superpenguinPresale

v0.8.24+commit.7dd6d404
v0.8.24

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

sepolia.etherscan.io

0xcad142cfc6dcc4e924e21d7c3927e4c282d7f797

[Disclaimer]

AuditBlock is not liable for any financial losses incurred due to its services. The information provided in this contract audit should not be considered financial advice. Please conduct your research to make informed decisions.

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.

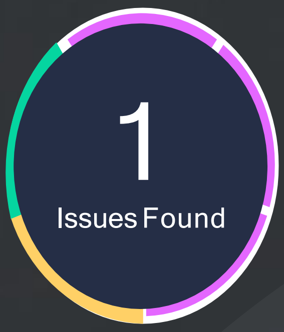
Gas Consumption

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	superpenguinPresale
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	1	0	0	0
Acknowledged Issues	0	0	0	0
Partially Resolved Issues	0	0	0	1
Resolved Issues	0	0	0	0

ID	File Name	Audit Status
10023	superpenguinPresale.sol	Medium

Smart Contract Weakness Classification (SWC) Vulnerabilities for Attacks

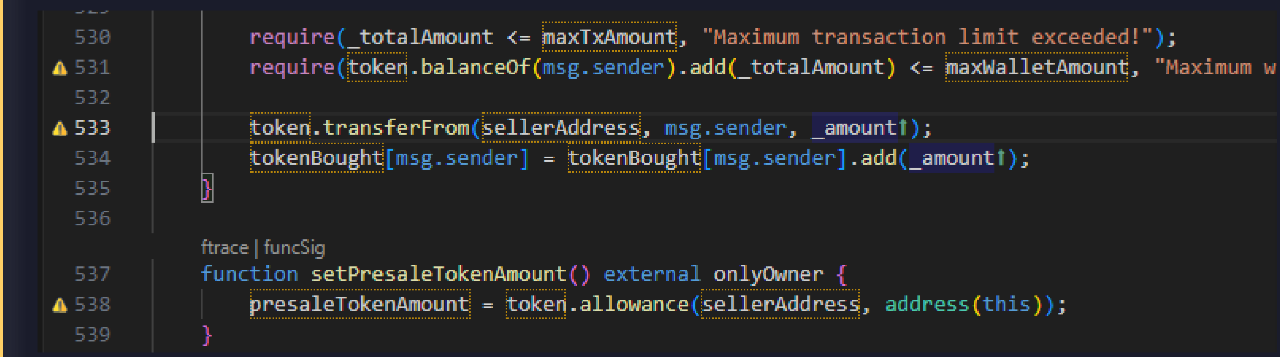
- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ Exception Disorder
- ✓ Gasless Send
- ✓ Use of tx.origin
- ✓ Compiler version not fixed
- ✓ Address hardcoded
- ✓ Divide before multiply
- ✓ Integer overflow/underflow
- ✓ Dangerous strict equalities

- ✓ 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

1. Unchecked transfer (ignores return value)



```
        require(_totalAmount <= maxTxAmount, "Maximum transaction limit exceeded!");
        require(token.balanceOf(msg.sender).add(_totalAmount) <= maxWalletAmount, "Maximum w

        token.transferFrom(sellerAddress, msg.sender, _amount);
        tokenBought[msg.sender] = tokenBought[msg.sender].add(_amount);
    }

    ftrace | funcSig
    function setPresaleTokenAmount() external onlyOwner {
        presaleTokenAmount = token.allowance(sellerAddress, address(this));
    }
```

Description

During the manual phase, our auditor identified several instances where methods ignored return values. Although this issue is categorized with high severity, we acknowledge that it may be part of the intended contract behavior in specific scenarios

Recommendation

It is important to ensure that you double-check your function's usability and how it's working with different behaviors. We recommend that you fix this by using Use SafeERC20, or by ensuring the transfer/transferFrom return value is checked.

Status

High

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

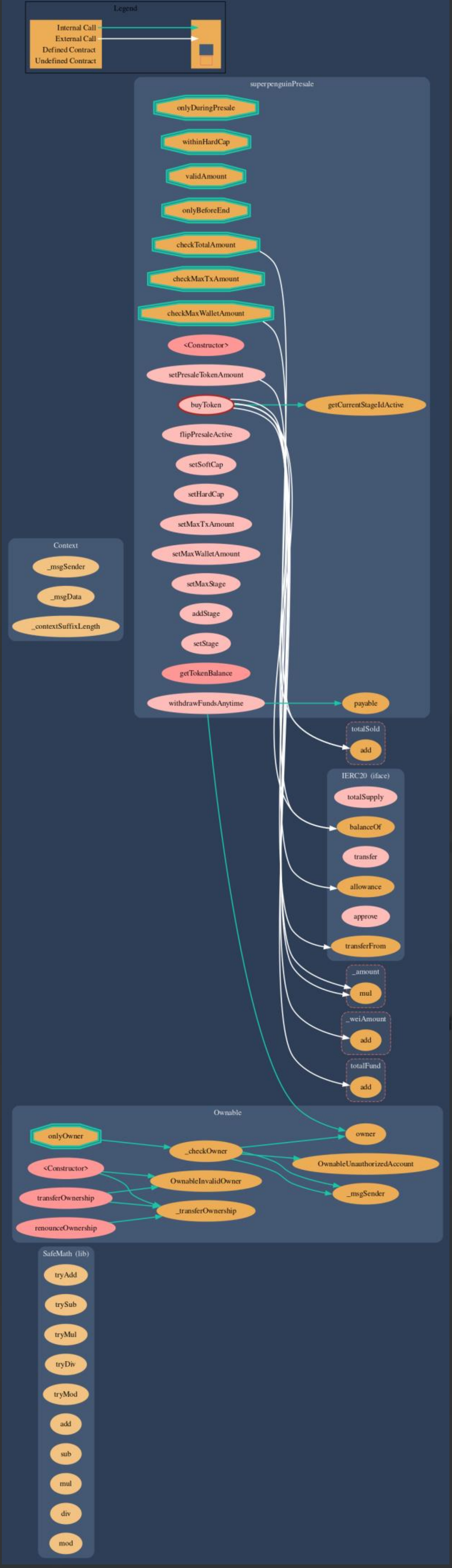
Informational Severity Issues

No issues found

Phase 2

```
 superpenguinPresale.buyToken(uint256) (contracts/superpenguinPresale.sol#514-535) uses arbitrary from
in transferFrom: token.transferFrom(sellerAddress,msg.sender,_amount)
(contracts/superpenguinPresale.sol#533)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#arbitrary-from-in-transferfrom

superpenguinPresale.buyToken(uint256) (contracts/superpenguinPresale.sol#514-535) ignores return value
by token.transferFrom(sellerAddress,msg.sender,_amount) (contracts/superpenguinPresale.sol#533)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

uperpenguinPresale.setMaxTxAmount(uint256) (contracts/superpenguinPresale.sol#553-555) should emit
an event for:
        - maxTxAmount = _amount (contracts/superpenguinPresale.sol#554)
superpenguinPresale.setMaxWalletAmount(uint256) (contracts/superpenguinPresale.sol#557-559) should
emit an event for:
        - maxWalletAmount = _amount (contracts/superpenguinPresale.sol#558)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

superpenguinPresale.constructor(address,address,address)._seller (contracts/superpenguinPresale.sol#509)
lacks a zero-check on :
                - sellerAddress = _seller (contracts/superpenguinPresale.sol#511)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
superpenguinPresale.buyToken(uint256) (contracts/superpenguinPresale.sol#514-535) uses timestamp for
comparisons
        Dangerous comparisons:
        - require(bool,string)(stage.start <= block.timestamp && block.timestamp <= stage.end,Presale period
invalid!) (contracts/superpenguinPresale.sol#517)
superpenguinPresale.getCurrentStageIdActive() (contracts/superpenguinPresale.sol#586-593) uses
timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp >= stages[i].start && block.timestamp <= stages[i].end
(contracts/superpenguinPresale.sol#588)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
Context._contextSuffixLength() (contracts/superpenguinPresale.sol#331-333) is never used and should be
removed
Context._msgData() (contracts/superpenguinPresale.sol#327-329) is never used and should be removed
SafeMath.div(uint256,uint256,string) (contracts/superpenguinPresale.sol#193-198) is never used and should
be removed
SafeMath.mod(uint256,uint256) (contracts/superpenguinPresale.sol#157-159) is never used and should be
removed
SafeMath.mod(uint256,uint256,string) (contracts/superpenguinPresale.sol#215-220) is never used and
should be removed
SafeMath.sub(uint256,uint256) (contracts/superpenguinPresale.sol#113-115) is never used and should be
removed
SafeMath.tryMul(uint256,uint256) (contracts/superpenguinPresale.sol#53-63) is never used and should be
removed
SafeMath.trySub(uint256,uint256) (contracts/superpenguinPresale.sol#41-46) is never used and should be
removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

superpenguinPresale.sellerAddress (contracts/superpenguinPresale.sol#450) should be immutable
superpenguinPresale.token (contracts/superpenguinPresale.sol#449) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-
declared-immutable
```

UML


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
superpenguinPresale
onlyDuringPresale
withinHardCap
validAmount
onlyBeforeEnd
checkTotalAmount
checkMaxTxAmount
checkMaxWalletAmount
<Constructor>
setPresaleTokenAmount
buyToken
getCurrentStageIdActive
flipPresaleActive
setSoftCap
setHardCap
setMaxTxAmount
setMaxWalletAmount
setMaxStage
addStage
setStage
getTokenBalance
withdrawFundsAnytime
payable
Context
_msgSender
_msgData
_contextSuffixLength
totalSold
add
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
_amount
mul
_weiAmount
add
totalFund
add
Ownable
onlyOwner
_checkOwner
owner
<Constructor>
OwnableUnauthorizedAccount
OwnableInvalidOwner
transferOwnership
_msgSender
_transferOwnership
renounceOwnership
SafeMath (lib)
tryAdd
trySub
tryMul
tryDiv
tryMod
add
sub
mul
div
mod

Closing Summary

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

Many issues were identified during the audit process, and their severity levels have been classified. Recommendations and best practices have also been provided to enhance code quality and security posture. The team has acknowledged all identified issues.

Disclaimer

AuditBlock does not provide security warranties, investment advice, or endorsements of any platform. This audit does not guarantee the security or correctness of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice. The authors are not liable for any decisions made based on the information in this document. Securing smart contracts is an ongoing process. A single audit is not sufficient. We recommend that the platform's development team implement a bug bounty program to encourage further analysis of the smart contract by other third parties

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/0AuditBlock