MEXTPresale

v0. 0.8.19+commit.7dd6d404

v0.8.19

✦ Low-Risk

Low-risk code

✦ Medium-Risk

Medium-risk code

✦ High-Risk

High-risk code

0x37c4d8fa0fde62a53dd2a61aff40c0b4eaff2ccb

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.

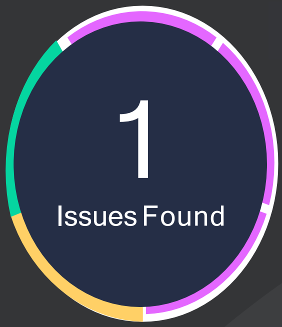
Gas Consumption

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	MEXTPresale
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	1
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	0	0	0

ID	File Name	Audit Status
10017	MEXTPresale.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

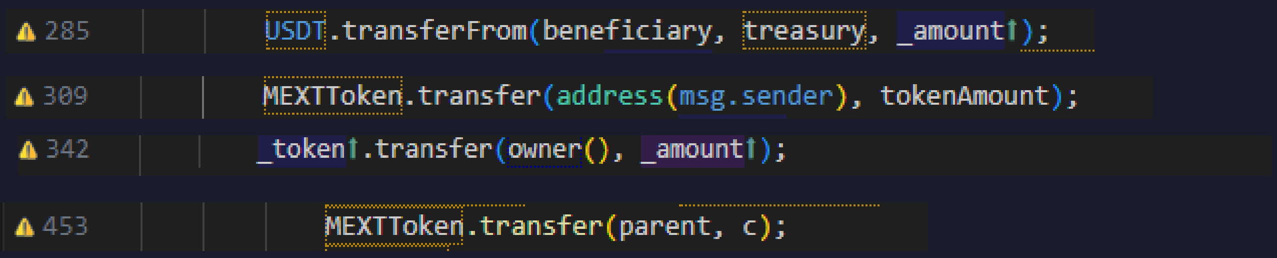
No issues found

Low Severity Issues

No issues found

Informational Severity Issues

1. unchecked transfer. (ignores return value)



```
USDT.transferFrom(beneficiary, treasury, _amount);
MEXTToken.transfer(address(msg.sender), tokenAmount);
_token.transfer(owner(), _amount);
    MEXTToken.transfer(parent, c);
```

Description

During the automation phase, our auditor identified several instances where methods ignore 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

Acknowledged

Phase 2

```
MEXTPresale.purchaseToken(uint256,address) (contracts/MEXTPresale.sol#282-311) ignores return value by
USDT.transferFrom(beneficiary,treasury,_amount) (contracts/MEXTPresale.sol#285)
MEXTPresale.purchaseToken(uint256,address) (contracts/MEXTPresale.sol#282-311) ignores return value by
MEXTToken.transfer(address(msg.sender),tokenAmount) (contracts/MEXTPresale.sol#309)
MEXTPresale.rescueTokens(IERC20,uint256) (contracts/MEXTPresale.sol#341-343) ignores return value by
_token.transfer(owner(),_amount) (contracts/MEXTPresale.sol#342)
MEXTPresale.payReferral(uint256) (contracts/MEXTPresale.sol#438-460) ignores return value by
MEXTToken.transfer(parent,c) (contracts/MEXTPresale.sol#453)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer


Reentrancy in MEXTPresale.payReferral(uint256) (contracts/MEXTPresale.sol#438-460):
	External calls:
	- MEXTToken.transfer(parent,c) (contracts/MEXTPresale.sol#453)
	State variables written after the call(s):
	- parentAccount.reward = parentAccount.reward.add(c) (contracts/MEXTPresale.sol#452)
	MEXTPresale.accounts (contracts/MEXTPresale.sol#236) can be used in cross function reentrancies:
	- MEXTPresale._preValidatePurchase(address,uint256,uint256) (contracts/MEXTPresale.sol#313-325)
	- MEXTPresale.accounts (contracts/MEXTPresale.sol#236)
	- MEXTPresale.addReferrer(address) (contracts/MEXTPresale.sol#399-431)
	- MEXTPresale.getlevelrefcount(address) (contracts/MEXTPresale.sol#362-364)
	- MEXTPresale.hasReferrer(address) (contracts/MEXTPresale.sol#368-370)
	- MEXTPresale.isCircularReference(address,address) (contracts/MEXTPresale.sol#372-392)
	- MEXTPresale.payReferral(uint256) (contracts/MEXTPresale.sol#438-460)
	- MEXTPresale.purchaseToken(uint256,address) (contracts/MEXTPresale.sol#282-311)
	- accounts[parent].levelRefCount[i] += 1 (contracts/MEXTPresale.sol#454)
	MEXTPresale.accounts (contracts/MEXTPresale.sol#236) can be used in cross function reentrancies:
	- MEXTPresale._preValidatePurchase(address,uint256,uint256) (contracts/MEXTPresale.sol#313-325)
	- MEXTPresale.accounts (contracts/MEXTPresale.sol#236)
	- MEXTPresale.addReferrer(address) (contracts/MEXTPresale.sol#399-431)
	- MEXTPresale.getlevelrefcount(address) (contracts/MEXTPresale.sol#362-364)
	- MEXTPresale.hasReferrer(address) (contracts/MEXTPresale.sol#368-370)
	- MEXTPresale.isCircularReference(address,address) (contracts/MEXTPresale.sol#372-392)
	- MEXTPresale.payReferral(uint256) (contracts/MEXTPresale.sol#438-460)
	- MEXTPresale.addReferrer(address) (contracts/MEXTPresale.sol#399-431)
	- MEXTPresale.getlevelrefcount(address) (contracts/MEXTPresale.sol#362-364)
	- MEXTPresale.hasReferrer(address) (contracts/MEXTPresale.sol#368-370)
	- MEXTPresale.isCircularReference(address,address) (contracts/MEXTPresale.sol#372-392)
	- MEXTPresale.payReferral(uint256) (contracts/MEXTPresale.sol#438-460)
	- MEXTPresale.purchaseToken(uint256,address) (contracts/MEXTPresale.sol#282-311)
	- accounts[msg.sender].totalUsdValue += _amount (contracts/MEXTPresale.sol#308)
	- MEXTPresale.purchaseToken(uint256,address) (contracts/MEXTPresale.sol#282-311)
Reentrancy in MEXTPresale.purchaseToken(uint256,address) (contracts/MEXTPresale.sol#282-311):
 MEXTPresale.totalSaleToken (contracts/MEXTPresale.sol#212) can be used in cross function reentrancies:
	- MEXTPresale._preValidatePurchase(address,uint256,uint256) (contracts/MEXTPresale.sol#313-325)
	- MEXTPresale.purchaseToken(uint256,address) (contracts/MEXTPresale.sol#282-311)
	- MEXTPresale.totalSaleToken (contracts/MEXTPresale.sol#212)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
```

```
MEXTPresale.setTotalTokenForSale(uint256) (contracts/MEXTPresale.sol#358-360) should emit an event for:
	- totalToken = _total (contracts/MEXTPresale.sol#359)
MEXTPresale.setMaxBuyLimit(uint256) (contracts/MEXTPresale.sol#462-464) should emit an event for:
	- max_buy_limit = _maxbuy (contracts/MEXTPresale.sol#463)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

MEXTPresale.setTreasuryWallet(address)._adr (contracts/MEXTPresale.sol#349) lacks a zero-check on :
		- treasury = _adr (contracts/MEXTPresale.sol#350)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
Reentrancy in MEXTPresale.purchaseToken(uint256,address) (contracts/MEXTPresale.sol#282-311):
	External calls:
	- USDT.transferFrom(beneficiary,treasury,_amount) (contracts/MEXTPresale.sol#285)
	- payReferral(tokenAmount) (contracts/MEXTPresale.sol#295)
		- MEXTToken.transfer(parent,c) (contracts/MEXTPresale.sol#453)
	State variables written after the call(s):
	- _raised += _amount (contracts/MEXTPresale.sol#304)
	- orders[++ latestOrderId] = OrderInfo(msg.sender,_amount,block.timestamp,tokenAmount)
(contracts/MEXTPresale.sol#298-303)
	- orderIds[msg.sender].push(latestOrderId) (contracts/MEXTPresale.sol#305)
	- orders[++ latestOrderId] = OrderInfo(msg.sender,_amount,block.timestamp,tokenAmount)
(contracts/MEXTPresale.sol#298-303)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
Reentrancy in MEXTPresale.purchaseToken(uint256,address) (contracts/MEXTPresale.sol#282-311):
	External calls:
	- USDT.transferFrom(beneficiary,treasury,_amount) (contracts/MEXTPresale.sol#285)
	- payReferral(tokenAmount) (contracts/MEXTPresale.sol#295)
		- MEXTToken.transfer(parent,c) (contracts/MEXTPresale.sol#453)
	- MEXTToken.transfer(address(msg.sender),tokenAmount) (contracts/MEXTPresale.sol#309)
	Event emitted after the call(s):
	- Purchase(tokenAmount,_amount,msg.sender,block.timestamp) (contracts/MEXTPresale.sol#310)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
Pragma version=0.8.19 (contracts/MEXTPresale.sol#7) necessitates a version too recent to be trusted.
Consider deploying with 0.8.18.
solc-0.8.19 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in MEXTPresale.rescueFunds() (contracts/MEXTPresale.sol#336-339):
	- (os) = address(owner()).call{value: address(this).balance}() (contracts/MEXTPresale.sol#337)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
 MEXTPresale.constructor() (contracts/MEXTPresale.sol#254-262) uses literals with too many digits:
	- tokenPrice = 1000000000000000000000000 (contracts/MEXTPresale.sol#257)
MEXTPresale.constructor() (contracts/MEXTPresale.sol#254-262) uses literals with too many digits:
	- totalToken = 100000000000000000000000 (contracts/MEXTPresale.sol#260)
MEXTPresale.constructor() (contracts/MEXTPresale.sol#254-262) uses literals with too many digits:
	- max_buy_limit = 10000000000000000000000 (contracts/MEXTPresale.sol#261)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
Loop condition i < refLevelRate.length (contracts/MEXTPresale.sol#379) should use cached array length
instead of referencing `length` member of the storage array.
 Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#cache-array-length
```

Closing Summary

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

One issue was 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://t.me/AuditBlock

https://github.com/AuditBlock

https://twitter.com/0AuditBlock