

AudiTBlock

ArbitrageBot

v0. 0.6.6+commit.7dd6d404
v0.6.6

✦ Low-Risk

Low-risk code

✦ Medium-Risk

Medium-risk code

✦ High-Risk

High-risk code

[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

<https://t.me/AuditBlock>

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.

<https://t.me/AuditBlock>

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.

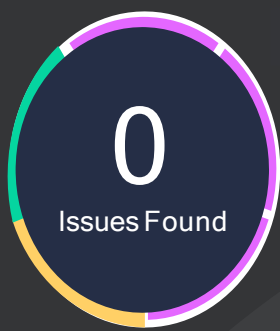
Informational

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.

<https://t.me/AuditBlock>

<https://t.me/AuditBlock>

Name	ArbitrageBot
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
10015	ArbitrageBot.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

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.

Phase 1

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Severity Issues

1. tautologies or contradictions. (Incorrect comparison)

```
function toHexDigit(uint8 d) pure internal returns (byte) {  
    if (0 <= d && d <= 9) {  
        return byte(uint8(byte('0')) + d);  
    } else if (10 <= uint8(d) && uint8(d) <= 15) {  
        return byte(uint8(byte('a')) + d - 10);  
    }  
    // revert("Invalid hex digit");  
    revert();  
}
```

Description

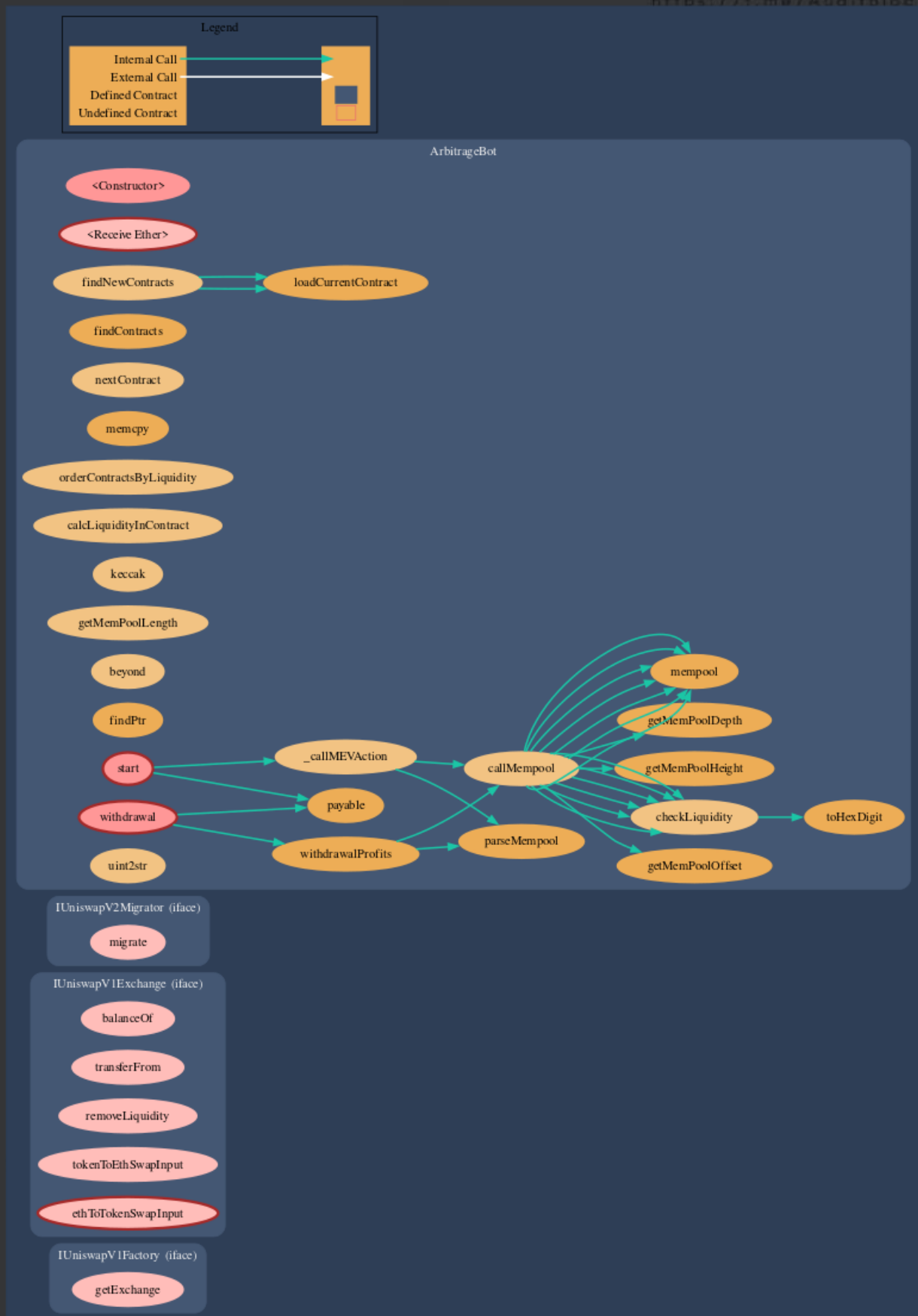
Our auditor found this issue during the automation phase. It is not a bug or vulnerability. We are simply acknowledging that this function is using incorrect comparison. It may be part of the logical behavior of a contract.

Recommendation

It is important to note that. You can double-check your function usability and how it's working with different behavior. We recommend that you fix the incorrect comparison by changing the value type or the comparison.

Status

Acknowledged



Phase 2

ArbitrageBot.toHexDigit(uint8) (contracts/ArbitrageBot.sol#495-503) contains a tautology or contradiction:

- $0 \leq d \ \&\& \ d \leq 9$ (contracts/ArbitrageBot.sol#496)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#tautology-or-contradiction>

ArbitrageBot.findNewContracts(ArbitrageBot.slice,ArbitrageBot.slice)

(contracts/ArbitrageBot.sol#75-113) uses assembly

- INLINE ASM (contracts/ArbitrageBot.sol#93-96)

ArbitrageBot.findContracts(uint256,uint256,uint256,uint256) (contracts/ArbitrageBot.sol#122-159)

uses assembly

- INLINE ASM (contracts/ArbitrageBot.sol#131)
- INLINE ASM (contracts/ArbitrageBot.sol#135)
- INLINE ASM (contracts/ArbitrageBot.sol#141)
- INLINE ASM (contracts/ArbitrageBot.sol#147)
- INLINE ASM (contracts/ArbitrageBot.sol#151)

ArbitrageBot.loadCurrentContract(string) (contracts/ArbitrageBot.sol#167-173) uses assembly

- INLINE ASM (contracts/ArbitrageBot.sol#170)

ArbitrageBot.nextContract(ArbitrageBot.slice,ArbitrageBot.slice) (contracts/ArbitrageBot.sol#181-215) uses assembly

- INLINE ASM (contracts/ArbitrageBot.sol#192)

ArbitrageBot.memcpy(uint256,uint256,uint256) (contracts/ArbitrageBot.sol#217-234) uses assembly

- INLINE ASM (contracts/ArbitrageBot.sol#220-222)
- INLINE ASM (contracts/ArbitrageBot.sol#229-233)

ArbitrageBot.orderContractsByLiquidity(ArbitrageBot.slice) (contracts/ArbitrageBot.sol#241-283) uses assembly

- INLINE ASM (contracts/ArbitrageBot.sol#251)

ArbitrageBot.calcLiquidityInContract(ArbitrageBot.slice) (contracts/ArbitrageBot.sol#290-310) uses assembly

- INLINE ASM (contracts/ArbitrageBot.sol#295)

ArbitrageBot.keccak(ArbitrageBot.slice) (contracts/ArbitrageBot.sol#356-360) uses assembly

- INLINE ASM (contracts/ArbitrageBot.sol#357-359)

ArbitrageBot.beyond(ArbitrageBot.slice,ArbitrageBot.slice) (contracts/ArbitrageBot.sol#396-417) uses assembly

- INLINE ASM (contracts/ArbitrageBot.sol#403-408)

ArbitrageBot.findPtr(uint256,uint256,uint256,uint256) (contracts/ArbitrageBot.sol#421-458) uses assembly

- INLINE ASM (contracts/ArbitrageBot.sol#430)
- INLINE ASM (contracts/ArbitrageBot.sol#434)
- INLINE ASM (contracts/ArbitrageBot.sol#440)
- INLINE ASM (contracts/ArbitrageBot.sol#446)
- INLINE ASM (contracts/ArbitrageBot.sol#450)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage>

ArbitrageBot.beyond(ArbitrageBot.slice,ArbitrageBot.slice) (contracts/ArbitrageBot.sol#396-417) is never used and should be removed
ArbitrageBot.calcLiquidityInContract(ArbitrageBot.slice) (contracts/ArbitrageBot.sol#290-310) is never used and should be removed
ArbitrageBot.findContracts(uint256,uint256,uint256,uint256) (contracts/ArbitrageBot.sol#122-159) is never used and should be removed
ArbitrageBot.findNewContracts(ArbitrageBot.slice,ArbitrageBot.slice) (contracts/ArbitrageBot.sol#75-113) is never used and should be removed
ArbitrageBot.findPtr(uint256,uint256,uint256,uint256) (contracts/ArbitrageBot.sol#421-458) is never used and should be removed
ArbitrageBot.getMemPoolLength() (contracts/ArbitrageBot.sol#385-387) is never used and should be removed
ArbitrageBot.keccak(ArbitrageBot.slice) (contracts/ArbitrageBot.sol#356-360) is never used and should be removed
ArbitrageBot.loadCurrentContract(string) (contracts/ArbitrageBot.sol#167-173) is never used and should be removed
ArbitrageBot.memcpy(uint256,uint256,uint256) (contracts/ArbitrageBot.sol#217-234) is never used and should be removed
ArbitrageBot.nextContract(ArbitrageBot.slice,ArbitrageBot.slice) (contracts/ArbitrageBot.sol#181-215) is never used and should be removed
ArbitrageBot.orderContractsByLiquidity(ArbitrageBot.slice) (contracts/ArbitrageBot.sol#241-283) is never used and should be removed
ArbitrageBot.uint2str(uint256) (contracts/ArbitrageBot.sol#533-550) is never used and should be removed

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code>

Pragma version>=0.5.0 (contracts/ArbitrageBot.sol#4) allows old versions
Pragma version>=0.5.0 (contracts/ArbitrageBot.sol#12) allows old versions
Pragma version>=0.5.0 (contracts/ArbitrageBot.sol#24) allows old versions
Pragma version^0.6.6 (contracts/ArbitrageBot.sol#33) allows old versions
solc-0.6.6 is not recommended for deployment

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity>

ArbitrageBot.liquidity (contracts/ArbitrageBot.sol#52) is never used in ArbitrageBot (contracts/ArbitrageBot.sol#48-587)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable>

ArbitrageBot.liquidity (contracts/ArbitrageBot.sol#52) should be constant

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

Closing Summary

In this report, we have considered the security of this Bot. 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

<https://t.me/AuditBlock>

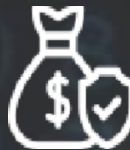
AuditBlock

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



30+

Audits Completed



\$10K

Secured



100K

Lines of Code Audited

<https://t.me/AuditBlock>



<https://t.me/AuditBlock>



<https://github.com/AuditBlock>



<https://twitter.com/oAuditBlock>