AudiTBlock

Sandwich Bot

v0.8.0

✦ Low-Risk

low-risk code

✦ Medium-Risk

medium-risk code

✦ High-Risk

high-risk code

Sandwich Bot

Executive Summary

Project Name Sandwich Bot

Overview This bot is a type of automated trading bot that exploits price slippage on decentralized exchanges (DEXs) like Uniswap. It works by placing a buy order for a token in a Uniswap pair and then placing a sell order for the same token in the same pair at a slightly higher price. The bot then cancels the buy order and takes the profit from the difference in prices.

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.

High Medium Low Informational

	High	Medium	Low	Informational
Open Issues	1	0	0	0
Acknowledged Issues	0	2	1	1
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	0	0	0

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

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.

Manual Testing

A. Project - Sandwich Bot

High Severity Issues

1. Unsecured code activities

Description

in the auditing process has found some unsecured code activities in your bot. This means that there is a vulnerability in the code of your wallet that could allow a bot to access your wallet completely. This could work as a vulnerability if they are not fixed

Remediation

Implement To prevent this issue, it is important to always Save your funds in other Wallet. Just keep trading funds in this account. Enable two-factor authentication (2FA). 2FA adds an extra layer of security to your wallet by requiring you to enter a code from your phone in addition to your password. Keep your bot up to date. Bot updates often include security patches that can help to protect your wallet from vulnerabilities.

Status

open

Medium Severity Issues

1.Certain unused functions

Description

Unused functions can be a problem because they can make your code less efficient. When the Solidity compiler compiles your code, it has to create bytecode for every function, even if the function is never called. This can add unnecessary size and complexity to your code.

Remediation

Be careful when defining functions. Only define functions that you are sure you will need.

Status

Acknowledged

3. Contract That Lock Ether

Description

The contract has no function to withdraw ethers, then it is impossible to withdraw the ether from the contract. This means that the user will lose their ether if they deposit or send directly into the contract.

Remediation

Implement To prevent this issue. if you are considering depositing ether into a contract, it is important to carefully Make sure that you understand why the contract has no function to directly withdraw ethers. Contract using just simple methods to receive Erc20 Token and make sure that you are comfortable with the risks involved.

Status

Acknowledged

4. Local or state variable.

Description

The return value of an external call in Solidity. In Solidity, when you call a function on another contract, the return value of the function is not stored in a local or state variable by default. This means that the return value of the function is lost after the function call has finished executing.

Remediation

It is important to note that not storing the return value of an external call can lead to security vulnerabilities. For example, if you call a function on another contract and do not store the return value, an attacker could modify the return value of the function and trick your contract into doing something malicious.

Status

Acknowledged

Automated Test

```
 Contract locking ether found:
        Contract Sandwich (contracts/Sandwich.sol#191-329) has payable
functions:
         - Sandwich.receive() (contracts/Sandwich.sol#204)
         - Sandwich.fallback() (contracts/Sandwich.sol#234-307)
        But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#contracts-that-lock-ether

Sandwich.sellToken(address,address,uint256) (contracts/Sandwich.sol#309-327)
ignores return value by (reserve0,reserve1) =
IUniswapV2Pair(pair).getReserves() (contracts/Sandwich.sol#313)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#unused-return

Function IUniswapV2Pair.DOMAIN_SEPARATOR() (contracts/Sandwich.sol#155) is
not in mixedCase
Function IUniswapV2Pair.PERMIT_TYPEHASH() (contracts/Sandwich.sol#156) is
not in mixedCase
Function IUniswapV2Pair.MINIMUM_LIQUIDITY() (contracts/Sandwich.sol#173) is
not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#conformance-to-solidity-naming-conventions

Sandwich.constructor(address)._owner (contracts/Sandwich.sol#206) lacks a
zero-check on :
                - user = _owner (contracts/Sandwich.sol#207)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#missing-zero-address-validation

Sandwich.fallback() (contracts/Sandwich.sol#234-307) uses assembly
        - INLINE ASM (contracts/Sandwich.sol#238-306)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#assembly-usage

Different versions of Solidity are used:
        - Version used: ['>=0.5.0', '>=0.8.0']
        - >=0.5.0 (contracts/Sandwich.sol#138)
        - >=0.8.0 (contracts/Sandwich.sol#7)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#different-pragma-directives-are-used

SafeTransfer.safeApprove(IERC20,address,uint256)
(contracts/Sandwich.sol#122-131) is never used and should be removed
SafeTransfer.safeTransferETH(address,uint256) (contracts/Sandwich.sol#133-
136) is never used and should be removed
SafeTransfer.safeTransferFrom(IERC20,address,address,uint256)
(contracts/Sandwich.sol#94-109) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#dead-code
```

```
 Low level call in SafeTransfer.safeTransferFrom(IERC20,address,address,uint256)
(contracts/Sandwich.sol#94-109):
        - (s) =
address(token).call(abi.encodeWithSelector(IERC20.transferFrom.selector,from,to,val
ue)) (contracts/Sandwich.sol#100-107)
Low level call in SafeTransfer.safeTransfer(IERC20,address,uint256)
(contracts/Sandwich.sol#111-120):
        - (s) =
address(token).call(abi.encodeWithSelector(IERC20.transfer.selector,to,value))
(contracts/Sandwich.sol#116-118)
Low level call in SafeTransfer.safeApprove(IERC20,address,uint256)
(contracts/Sandwich.sol#122-131):
        - (s) =
address(token).call(abi.encodeWithSelector(IERC20.approve.selector,to,value))
(contracts/Sandwich.sol#127-129)
Low level call in SafeTransfer.safeTransferETH(address,uint256)
(contracts/Sandwich.sol#133-136):
        - (s) = to.call{value: value}(new bytes(0)) (contracts/Sandwich.sol#134)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-
calls
```

Functional Testing

Some of the tests performed are mentioned below:

- ✔ Should revert when non-owner calls the sellToken function
- ✔ Should be able to start bot activities after the bot owner successfully calls the functions
- ✔ Should revert when users call Recover ERC20 when the bot has not started
- ✔ Should be able to Recover ERC20 successfully and continuously.
- ✔ Should revert if the none -owner user attempts SafeTransfer.safeTransferETH
- ✔ Should be able to start the bot with IUniswapV2Pair.DOMAIN_SEPARATOR()
- ✘ Should be able to stop bot by contract owner when the market is unstable
- ✘ withdraw funds from the contracts without any hindrance.

SOLDITIY TESTING

Contracts/Sandwich.sol

```
  7:1   warning  Found more than One contract per file. 4 contracts found!              one-contract-per-file
 100:22 warning  Avoid to use low level calls                                          avoid-low-level-calls
 108:9  warning  Use Custom Errors instead of require statements                       custom-errors
 116:22 warning  Avoid to use low level calls                                          avoid-low-level-calls
 119:9  warning  Use Custom Errors instead of require statements                       custom-errors
 127:22 warning  Avoid to use low level calls                                          avoid-low-level-calls
 130:9  warning  Use Custom Errors instead of require statements                       custom-errors
 135:9  warning  Use Custom Errors instead of require statements                       custom-errors
 138:1  error    Compiler version >=0.5.0 does not satisfy the ^0.8.0 semver requirement  compiler-version
 141:68 warning  Rule is set with explicit type [var/s: uint]                          explicit-types
 142:62 warning  Rule is set with explicit type [var/s: uint]                          explicit-types
 147:51 warning  Rule is set with explicit type [var/s: uint]                          explicit-types
 148:62 warning  Rule is set with explicit type [var/s: uint]                          explicit-types
 149:79 warning  Rule is set with explicit type [var/s: uint]                          explicit-types
 151:39 warning  Rule is set with explicit type [var/s: uint]                          explicit-types
 152:35 warning  Rule is set with explicit type [var/s: uint]                          explicit-types
 153:53 warning  Rule is set with explicit type [var/s: uint]                          explicit-types
 155:5  warning  Function name must be in mixedCase                                    func-name-mixedcase
 156:5  warning  Function name must be in mixedCase                                    func-name-mixedcase
 157:59 warning  Rule is set with explicit type [var/s: uint]                          explicit-types
 159:53 warning  Rule is set with explicit type [var/s: uint]                          explicit-types
 159:65 warning  Rule is set with explicit type [var/s: uint]                          explicit-types
 161:40 warning  Rule is set with explicit type [var/s: uint]                          explicit-types
 161:54 warning  Rule is set with explicit type [var/s: uint]                          explicit-types
 162:40 warning  Rule is set with explicit type [var/s: uint]                          explicit-types
 162:54 warning  Rule is set with explicit type [var/s: uint]                          explicit-types
 165:9  warning  Rule is set with explicit type [var/s: uint]                          explicit-types
 166:9  warning  Rule is set with explicit type [var/s: uint]                          explicit-types
 167:9  warning  Rule is set with explicit type [var/s: uint]                          explicit-types
 173:5  warning  Function name must be in mixedCase                                    func-name-mixedcase
 173:57 warning  Rule is set with explicit type [var/s: uint]                          explicit-types
 178:60 warning  Rule is set with explicit type [var/s: uint]                          explicit-types
 179:60 warning  Rule is set with explicit type [var/s: uint]                          explicit-types
 180:45 warning  Rule is set with explicit type [var/s: uint]                          explicit-types
 182:49 warning  Rule is set with explicit type [var/s: uint]                          explicit-types
 183:49 warning  Rule is set with explicit type [var/s: uint]                          explicit-types
 183:63 warning  Rule is set with explicit type [var/s: uint]                          explicit-types
 184:19 warning  Rule is set with explicit type [var/s: uint]                          explicit-types
 184:36 warning  Rule is set with explicit type [var/s: uint]                          explicit-types
 195:5  warning  Immutable variables name are set to be in capitalized SNAKE_CASE      immutable-vars-naming
 212:9  warning  Use Custom Errors instead of require statements                       custom-errors
 234:5  warning  Fallback function must be simple                                      no-complex-fallback
 238:9  warning  Avoid to use inline assembly. It is acceptable only in rare cases     no-inline-assembly
 310:9  warning  Use Custom Errors instead of require statements                       custom-errors
 315:13 warning  Rule is set with explicit type [var/s: uint]                          explicit-types
 316:13 warning  Rule is set with explicit type [var/s: uint]                          explicit-types
 317:13 warning  Rule is set with explicit type [var/s: uint]                          explicit-types
 321:13 warning  Rule is set with explicit type [var/s: uint]                          explicit-types
 322:13 warning  Rule is set with explicit type [var/s: uint]                          explicit-types
 323:13 warning  Rule is set with explicit type [var/s: uint]                          explicit-types
```

SWC Attacks

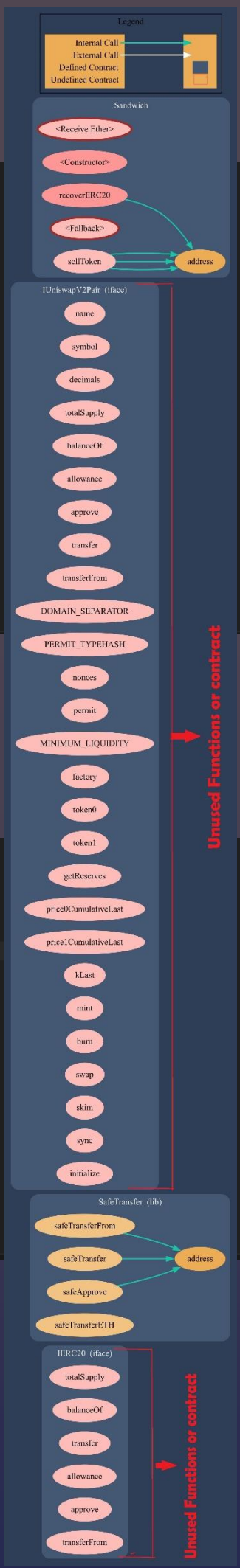
ID	Title		Test Result
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	✔
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	✔
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	✔
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	✔
SWC-127	Arbitrary Jump with Function TypeVariable	CWE-695: Use of Low-Level Functionality	#
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	✔
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	✔
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	✔

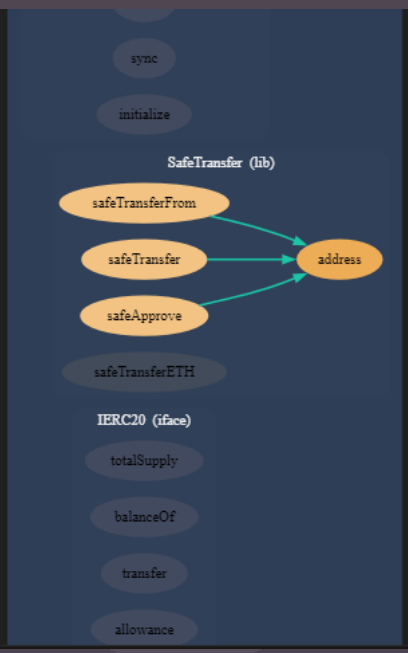
ID	Title		Test Result
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	✔
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✔
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	✔
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	✔
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	✔
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	$
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	✔
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	$
SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	✔
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	$

Solidity Call graph

UnitTest stub | dependencies | uml | draw.io

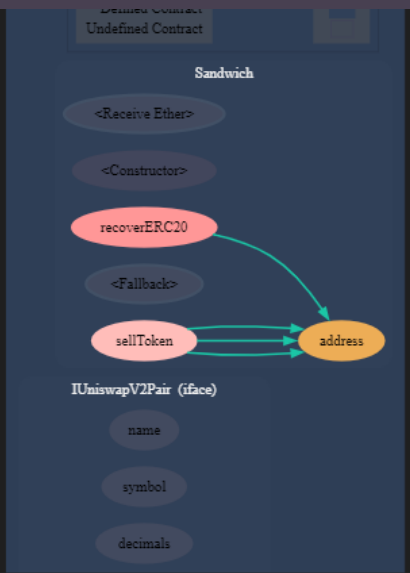
```
library SafeTransfer {
    ftrace | funcSig
    function safeTransferFrom(
        IERC20 token,
        address from,
        address to,
        uint256 value
    ) internal {
        (bool s, ) = address(token).call(
            abi.encodeWithSelector(
                IERC20.transferFrom.selector,
                from,
                to,
                value
            )
        );
        require(s, "safeTransferFrom failed");
    }

    ftrace | funcSig
    function safeTransfer(
        IERC20 token,
        address to,
        uint256 value
```





UnitTest stub | dependencies | uml | draw.io

```
contract Sandwich {
    using SafeTransfer for IERC20;

    // Authorized
    address internal immutable user;

    // transfer(address,uint256)
    bytes4 internal constant ERC20_TRANSFER_ID = 0xa9059cbb;

    // swap(uint256,uint256,address,bytes)
    bytes4 internal constant PAIR_SWAP_ID = 0x022c0d9f;

    // Contructor sets the only user
    ftrace
    receive() external payable {}

    ftrace
    constructor(address _owner) {
        user = _owner;
    }
```



Closing Summary

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

Some issues of Medium, Low, and Informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture. In the end, Sandwich Bot. Team Acknowledged all Issues.

Disclaimer

AuditBlock smart contract audit is not a security warranty, investment advice, or an endorsement of this Platform. This audit does not provide a security or correctness guarantee for the audited smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

Securing smart contracts is a multi-step process. One audit cannot be considered enough. We recommend that the contract Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

About AudiTBlock

Contractsaudit is a secure smart contracts audit platform designed by Auditblock We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

300+
Audits Completed

$3B
Secured

300K
Lines of Code Audited

Audit Report,
3 September 2023

For
Sandwich Bot

auditblock@gmail.com

AudiTBlock