QuillAudits

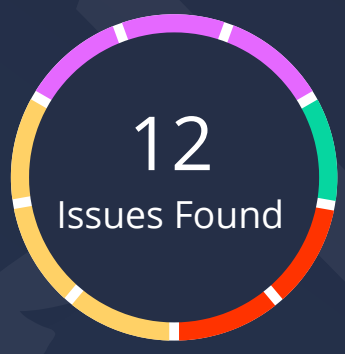
Audit Report
April, 2023

For

Table of Content

Executive Summary

Project Name	Advon
Overview	A Defi token with long term customer focused tokenomics
Timeline	27th March, 2023 to 5th April, 2023
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyze Advon codebase for quality, security, and correctness. *https://drive.google.com/file/d/1liqPLCpazbdn1ZTRXqZOiNchVeOfUX48/view?usp=share_link* Contracts in Scope: Advon.sol
Fixed In	*https://drive.google.com/file/d/1fR2BmYZr501kJwbUCryV5HeDFkiADTxB/view?usp=share_link*



High
Medium
Low
Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	1	1	1	3
Partially Resolved Issues	2	3	0	0
Resolved Issues	0	0	0	1

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.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Checked Vulnerabilities

- 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
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

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

Structural Analysis

In this step, we have analysed 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 analysed, 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 behaviour 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

High Severity Issues

1. Centralization Risk

Description

The contract owner can adjust the tax to as high as 100% on transactions, limit the tradeable limit of each user's balance. The owner account should be taken care to not be mismanaged else the privileges associated with the account can slip into the hands of the hacker. Also, if renouncing ownership is not done appropriately, the current contract owner can renounce ownership, and lose access to owner privileges. Functions like setting tax, and wallet addresses will be uncallable and any updates needed to be made will be inaccessible.

Remediation

Carefully manage the owner account by using a multi-signature wallet or implementing community governance to ensure that updates to the project happen by some form of consensus. A timelock can be implemented between changes as well.
The renounceOwnership() and transferOwnership() can be set up for 2-step verification to prevent mistaken privilege transfer or renouncing.

References

- Link 1
- Link 2

Status

Acknowledged

2. Frontrunning

Description

Upon deployment, the expected contract owner would have to call initialize( ) else the owner characteristics and privileges do not get set. This is an issue for two reasons:

Scenario 1: Initialize is frontrun by a malicious actor
If anyone asides from the intended owner calls initialize, they automatically become the owner of the Advon contract with the privileges of the owner address. Bots could be listening on the network for scenarios set up just like this and take advantage of the exploit opportunity.

Scenario 2: Initialize does not get called by anyone
This would make the owner variable set to address(0), and core functionality will be unusable since address(0) currently does not have a signer and cannot pass the onlyOwner check.

Remediation

With the current contract structure, it is recommended to call the functions in this order to mitigate risk:
- deploy contract
- call initialize( )
- set wallets, release time, etc to ensure funds are sent to the right address and proper timing follows.
- call distribute( )

The contract deployer has to be aware of safe deployment practices, they can refer to OpenZeppelin's guide linked: *Writing Upgradeable Contracts - OpenZeppelin Docs.*

Status

Partially Resolved

Client's Remarks: The contract when deployed via the proxy avoids this issue of frontrunning by a malicious actor.

3. Token allocation and transfers

Description

The ecosystemFund address is never set and if left unchecked would lead to loss of funds if the tokens (in the contract itself) are sent out to the ecosystemAddress which is the dead address perpetually. 50% of the supply is minted to the contract as well not sent to the respective addresses as well. A compromise of the contract will automatically put 50% of the total supply at risk.
The dev also described eligibleWallets as those that do not need to go through the usual phase of locking and keeping tokens like normal accounts do. They could have the privilege of moving tokens without a fee or lockup period but they do not. The only privilege available to the eligibileWallets in the codebase is the ability to call transfer() and transferFrom() tokens before the saleEndTime.
In the most recent inclusion, when transfers are made it first checks if the address is an eligible wallet, and also if it is an eligible reseller. If an address is an eligible reseller, it should not deduct any tokens to be locked when tokens pass through it. The flaw here is that the eligible reseller check the 'to' address and not the 'owner' address else it could have some tokens locked like every other non-privileged account.
If the batch transfer function is called with an eligible reseller's address as well, it would lead to same deduction for locked tokens which should not happen with privileged accounts.

```
function transferFrom(address from, address to, uint256 amount) public ... {
    if(eligibleWallets[owner]) {
      if(eligibleResellers[owner]) { // should check 'to' not owner here
        _transfer(owner,to,amount);
       }
           }
}
```

Remediation

Provide a setEcosystemAddress() function to update the ecosystem address as needed. Also, mint tokens to the already specified addresses to reduce the scale of risk in the event of a compromise. For eligible wallets, ensure the implementation matches the dev description of expected functionality. Ensure proper implementation for the eligible resellers checks in transfer, transferFrom, and batchTransfer. For batchTransfer, include checks for privileged accounts.

Status

Partially Resolved

Medium Severity Issues

4. Early token unlocks

Description

The releaseTime is checked to confirm it's not zero but does not check to see if releaseTime is greater than contractCreatedTime.

```
function setRealeaseTime(uint256 _time) external onlyOwner{
    require(_time != 0, "Should not be Zero");
    releaseTime = _time;
}
```

This would mean the claim function can be manipulated to dispense more tokens much earlier than expected if the owner sets it that way.
Also, salesEndTime can be manipulated in the same way to pass the check:

```
require(block.timestamp > contractCreatedTime + saleEndTime, ...);
```

Remediation

Ensure the arithmetic checks are not easily manipulated to avoid privileged users gaming the system.

Status

Acknowledged

5. Null token transfers

Description

On token transfers, the arithmetic calculations carried out determine how many tokens will be locked and how many can be transferred to users. Although the checks are implemented, the transfers from the caller to the msg.sender and the token do not happen

Remediation

Ensure the arithmetic checks are accurate, and tokens are transferred from the caller to the appropriate receivers.

Status

Partially Resolved

6. Input sanitization/Parity checks

Description

Although underflows and overflows are checked internally in solidity versions >0.8.0, multiple logical arithmetic checks would be needed in the codebase dependent on the client. setReleaseTime(), setSupply_Team_Allocation(), setBuyandSellTax(), setTokenPrice() are likely pivots to exploit this vulnerability.
When eligible resellers are added, the NumberofResellers variable is incremented by 1. The function does not check if the address to be added is already an eligibleReseller so the NumberofResellers variable can be incremented to the limit of uint256. Due to the capacity of uint256 it is unlikely to result in a denial of service but it still is an issue of improper accounting.

Remediation

- setReleaseTime() has been described in the exploit scenario above
- setSupply_Team_Allocation() can be done with hardcoded values because the whitepaper clearly specifies how much will be allocated to Founder, Marketing, Ecosystem, and Staking,
- setBuyandSellTax() if implemented properly could lead to users swapping and losing all their tokens because there is currently no cap to how much tax would be deducted per transaction,
- setTokenPrice() can be updated to any value without prior notice by the owner but with the values corresponding to the epoch stated in the whitepaper a struct can be implemented to avoid mistaken allocations of price by the owner.
- setProcessingFee() and setNetworkingFee() also should be checked, updated or removed if not in use.
- Add checks for address existence before updating

Status

Acknowledged

7. Missing test cases

Description

The codebase lacks unit test coverage. It is advisable to have test coverage greater than 95% of the codebase to reduce unexpected functionality and help fuzz test as many invariants as possible.

Remediation

Include unit tests for the codebase.

Status

Partially Resolved

Low Severity Issues

8. Unused code

Description

The contract declares IPancakeFactory, IPancakeRouter, and IPancakePair but they are not implemented in any form throughout the scope of the codebase. TradeableLimit, ProcessingFee, and NetworkingFee are declared but not implemented as well. The dev's comment implied that tradeLimit allows only a certain percentage of the user's tokens to be tradeable at a certain time. The current version of the codebase does not put this functionality to use.

Remediation

Consider cleaning up the contract by removing the unused interfaces, implementing said functionality, and deleting dead code to help reduce the contract size and aid deployment.

Status

Acknowledged

Informational Issues

9. Event emission

Description

Changes to state in deployed contracts can be easily tracked using events, especially when logs would need to be kept. The contract allows multiple key variables to be reset and it would be advisable to emit events when this occurs.

Remediation

Emit events and index them (where needed) to keep track of critical changes and aid searches for functions that update state.

Status

Acknowledged

10. Unlocked pragma

Description

The solidity pragma version in this codebase is unlocked.

Remediation

It is advised to use a specific Solidity version when deploying to production to reduce the surface of attacks with future releases which were not accounted for when the contracts originally went live.

Status

Acknowledged

11. Unnecessary access control

Description

The internal “_mint()” function is only called by the distribute() function which has the onlyOwner modifier attached to it as well. Calling it again internally will lead to extra costs spent on gas which can be avoided.

Remediation

Consider removing the onlyOwner modifier in the “_mint” function.

Status

Acknowledged

12. Low code and documentation readability

Description

The whitepaper only accounts for 95% of the token supply, there is an unspecified 5% allocation. The codebase contains many typos, and does not conform to the recommended Solidity standard of code formatting.

The error message for transfers is inaccurate. Due to the variable nature of saleEndTime, the message ‘after one year’ can be wrong if saleEndTime is shorter or longer.

```
require(block.timestamp>contractCreatedTime+saleEndTime,"only can transfer after one year")
```

Remediation

Ensure that the contract code matches the documentation provided. To improve the ease of reading the codebase, it can be passed through a spellcheck and the tool “Prettier” can be used on the contract to aid formatting.

Status

Resolved

General Recommendations

- Reduce frontrunning risk by performing all function calls in the single deployment / initialize step. Addresses can be assigned during contract creation and set as constant or immutable to reduce contract size.
- Remove redundant variables, e.g. tradeLimit and tradeablePercent if they point to the same thing.
- Pack storage for less accessed storage variables

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

```
Advon.withdraw(address,uint256) (contracts/advon.sol#1340-1344) ignores return value by IERC20Upgradeable(_erc20Address).transfer(address(msg.send
mount) (contracts/advon.sol#1343)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

Advon.ecosystemFund (contracts/advon.sol#606) is never initialized. It is used in:
        - Advon.claim() (contracts/advon.sol#732-805)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables

Advon.setstakingAndTrustWallet(address) (contracts/advon.sol#840-843) should emit an event for:
        - stakingAndTrust = _wallet (contracts/advon.sol#842)
Advon.setFoundersWallet(address) (contracts/advon.sol#845-848) should emit an event for:
        - teamAndFounders = _wallet (contracts/advon.sol#847)
Advon.setmarketingWallet(address) (contracts/advon.sol#850-853) should emit an event for:
        - marketing = _wallet (contracts/advon.sol#852)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-access-control

Advon.setSaleEndTime(uint256) (contracts/advon.sol#946-949) should emit an event for:
        - saleEndTime = _time (contracts/advon.sol#948)
Advon.setTradeablebalance(uint256) (contracts/advon.sol#951-953) should emit an event for:
        - tradeablePercent = _percent (contracts/advon.sol#952)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

Advon.setReserveWallet(address)._wallet (contracts/advon.sol#722) lacks a zero-check on :
                - reserve = _wallet (contracts/advon.sol#723)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Advon.claim() (contracts/advon.sol#732-805) uses timestamp for comparisons
        Dangerous comparisons:
        - monthsPassed > 12 (contracts/advon.sol#737)
        - require(bool,string)(block.timestamp >= releaseTime + monthTime,Lock Peroid is not over) (contracts/advon.sol#741)
        - require(bool,string)(block.timestamp - FounderlastClaimedTime >= monthTime,only can claim once in a month) (contracts/advon.sol#742)
        - require(bool,string)(FoundermonthsClaimed < 12,max Unlock is reached) (contracts/advon.sol#743)
        - require(bool)(transferrableTokens <= 8000000 * 10 ** decimals()) (contracts/advon.sol#748)
        - monthsPassed_scope_0 > 12 (contracts/advon.sol#757)
        - require(bool,string)(block.timestamp >= (contractCreatedTime + monthTime),Lock Peroid is not over) (contracts/advon.sol#760)
        - monthsPassed_scope_0 > 12 (contracts/advon.sol#764)
        - require(bool,string)(MarketingmonthsClaimed < 12,max Unlock is reached) (contracts/advon.sol#767)
        - require(bool)(transferrableTokens_scope_2 <= 5000000 * 10 ** decimals()) (contracts/advon.sol#771)
        - monthsPassed_scope_0 > 12 (contracts/advon.sol#777)
        - require(bool,string)(ecosystemmonthsClaimed < 12,max Unlock is reached) (contracts/advon.sol#780)
        - require(bool)(transferrableTokens_scope_4 <= 5000000 * 10 ** decimals()) (contracts/advon.sol#784)
        - monthsPassed_scope_0 > 12 (contracts/advon.sol#790)
        - require(bool,string)(stakingAndTrustmonthsClaimed < 12,max Unlock is reached) (contracts/advon.sol#793)
        - require(bool)(transferrableTokens_scope_6 <= 5000000 * 10 ** decimals()) (contracts/advon.sol#797)
Advon.transfer(address,uint256) (contracts/advon.sol#969-1001) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp > contractCreatedTime + saleEndTime,only can transfer after the sale End) (contracts/advon.sol#985)
Advon.getLockedBalance(address) (contracts/advon.sol#1003-1015) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool)(block.timestamp <= (boughtTime[msg.sender][i] + yearTime)) (contracts/advon.sol#1010)
Advon.batchTransfer(address[],uint256[]) (contracts/advon.sol#1026-1067) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp > (contractCreatedTime + saleEndTime),only can transfer after one year) (contracts/advon.sol#1047)
Advon.claimLockedBalances(uint256) (contracts/advon.sol#1070-1087) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool)(block.timestamp <= (boughtTime[msg.sender][i] + yearTime)) (contracts/advon.sol#1079)
Advon.transferFrom(address,address,uint256) (contracts/advon.sol#1128-1160) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp > contractCreatedTime + yearTime,only can transfer after one year) (contracts/advon.sol#1145)
Advon.Burn(address,uint256) (contracts/advon.sol#1186-1208) uses timestamp for comparisons
        Dangerous comparisons:
        - yearspassed < 1 (contracts/advon.sol#1189)
        - yearspassed > 1 && yearspassed < 2 (contracts/advon.sol#1194)
        - require(bool,string)(block.timestamp <= (contractCreatedTime + (yearTime * 2)),2nd year ) (contracts/advon.sol#1195)
        - yearspassed > 2 && yearspassed < 3 (contracts/advon.sol#1198)
        - require(bool,string)(block.timestamp <= (contractCreatedTime + (yearTime * 3)),3rd year ) (contracts/advon.sol#1199)
        - yearspassed > 3 && yearspassed < 4 (contracts/advon.sol#1202)
        - require(bool,string)(block.timestamp <= (contractCreatedTime + (yearTime * 4)),4th year ) (contracts/advon.sol#1203)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

AddressUpgradeable._revert(bytes,string) (contracts/advon.sol#319-331) uses assembly
        - INLINE ASM (contracts/advon.sol#324-327)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

```
AddressUpgradeable._revert(bytes,string) (contracts/advon.sol#319-331) is never used and should be removed
AddressUpgradeable.functionCall(address,bytes) (contracts/advon.sol#198-200) is never used and should be removed
AddressUpgradeable.functionCall(address,bytes,string) (contracts/advon.sol#208-214) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256) (contracts/advon.sol#227-233) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (contracts/advon.sol#241-250) is never used and should be removed
AddressUpgradeable.functionStaticCall(address,bytes) (contracts/advon.sol#258-260) is never used and should be removed
AddressUpgradeable.functionStaticCall(address,bytes,string) (contracts/advon.sol#268-275) is never used and should be removed
AddressUpgradeable.sendValue(address,uint256) (contracts/advon.sol#173-178) is never used and should be removed
AddressUpgradeable.verifyCallResult(bool,bytes,string) (contracts/advon.sol#307-317) is never used and should be removed
AddressUpgradeable.verifyCallResultFromTarget(address,bool,bytes,string) (contracts/advon.sol#283-299) is never used and should be removed
ContextUpgradeable.__Context_init() (contracts/advon.sol#504-505) is never used and should be removed
ContextUpgradeable.__Context_init_unchained() (contracts/advon.sol#507-508) is never used and should be removed
ContextUpgradeable._msgData() (contracts/advon.sol#513-515) is never used and should be removed
Initializable._getInitializedVersion() (contracts/advon.sol#481-483) is never used and should be removed
Initializable._isInitializing() (contracts/advon.sol#488-490) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (contracts/advon.sol#3) allows old versions
solc-0.8.19 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in AddressUpgradeable.sendValue(address,uint256) (contracts/advon.sol#173-178):
        - (success) = recipient.call{value: amount}() (contracts/advon.sol#176)
Low level call in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (contracts/advon.sol#241-250):
        - (success,returndata) = target.call{value: value}(data) (contracts/advon.sol#248)
Low level call in AddressUpgradeable.functionStaticCall(address,bytes,string) (contracts/advon.sol#268-275):
        - (success,returndata) = target.staticcall(data) (contracts/advon.sol#273)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Function ContextUpgradeable.__Context_init() (contracts/advon.sol#504-505) is not in mixedCase
Function ContextUpgradeable.__Context_init_unchained() (contracts/advon.sol#507-508) is not in mixedCase
Variable ContextUpgradeable.__gap (contracts/advon.sol#522) is not in mixedCase
Function IPancakeRouter.WETH() (contracts/advon.sol#538) is not in mixedCase
Event AdvoneligibleResellerAdded(address) (contracts/advon.sol#588) is not in CapWords
Function Advon.__ERC20_init(string,string) (contracts/advon.sol#695-699) is not in mixedCase
Function Advon.__ERC20_init_unchained(string,string) (contracts/advon.sol#701-705) is not in mixedCase
Parameter Advon.setRealeaseTime(uint256)._time (contracts/advon.sol#711) is not in mixedCase
Parameter Advon.setPrivateSaleWallets(address[])._wallets (contracts/advon.sol#716) is not in mixedCase
Parameter Advon.setReserveWallet(address)._wallet (contracts/advon.sol#722) is not in mixedCase
Parameter Advon.addeligibleWallet(address)._wallet (contracts/advon.sol#726) is not in mixedCase
Function Advon.setSupply_Team_Allocation(uint256,uint256,uint256,uint256) (contracts/advon.sol#825-831) is not in mixedCase
Parameter Advon.setSupply_Team_Allocation(uint256,uint256,uint256,uint256)._founders (contracts/advon.sol#825) is not in mixedCase
Parameter Advon.setSupply_Team_Allocation(uint256,uint256,uint256,uint256)._marketing (contracts/advon.sol#825) is not in mixedCase
Parameter Advon.setSupply_Team_Allocation(uint256,uint256,uint256,uint256)._ecosystem (contracts/advon.sol#825) is not in mixedCase
Parameter Advon.setSupply_Team_Allocation(uint256,uint256,uint256,uint256)._staking (contracts/advon.sol#825) is not in mixedCase
Parameter Advon.setSeedRoundWallet(address)._wallet (contracts/advon.sol#835) is not in mixedCase
Parameter Advon.setstakingAndTrustWallet(address)._wallet (contracts/advon.sol#840) is not in mixedCase
Parameter Advon.setFoundersWallet(address)._wallet (contracts/advon.sol#845) is not in mixedCase
Parameter Advon.setmarketingWallet(address)._wallet (contracts/advon.sol#850) is not in mixedCase
Parameter Advon.setLiquidityWallet(address)._wallet (contracts/advon.sol#855) is not in mixedCase
Parameter Advon.setTeamAndAdvisorsWallet(address)._wallet (contracts/advon.sol#860) is not in mixedCase
Parameter Advon.setBuyBackWallet(address)._wallet (contracts/advon.sol#867) is not in mixedCase
Function Advon.Getowner() (contracts/advon.sol#881-883) is not in mixedCase
Parameter Advon.iseligibleWallets(address)._wallet (contracts/advon.sol#924) is not in mixedCase
Parameter Advon.setTradeableLimit(uint256)._limitPercent (contracts/advon.sol#928) is not in mixedCase
Parameter Advon.setTokenPrice(uint256)._price (contracts/advon.sol#932) is not in mixedCase
Parameter Advon.setProcessingFee(uint256)._fee (contracts/advon.sol#938) is not in mixedCase
Parameter Advon.setNetworkingFee(uint256)._fee (contracts/advon.sol#942) is not in mixedCase
Parameter Advon.setSaleEndTime(uint256)._time (contracts/advon.sol#946) is not in mixedCase
Parameter Advon.setTradeablebalance(uint256)._percent (contracts/advon.sol#951) is not in mixedCase
Parameter Advon.addEligibleReseller(address)._wallet (contracts/advon.sol#955) is not in mixedCase
Parameter Advon.getLockedBalance(address)._wallet (contracts/advon.sol#1003) is not in mixedCase
Parameter Advon.getTradeableBalance(address)._wallet (contracts/advon.sol#1017) is not in mixedCase
Function Advon.Burn(address,uint256) (contracts/advon.sol#1186-1200) is not in mixedCase
Parameter Advon.Burn(address,uint256)._account (contracts/advon.sol#1186) is not in mixedCase
Parameter Advon.Burn(address,uint256)._value (contracts/advon.sol#1186) is not in mixedCase
Parameter Advon.withdraw(address,uint256)._erc20Address (contracts/advon.sol#1340) is not in mixedCase
Variable Advon.__gap (contracts/advon.sol#1422) is not in mixedCase
Variable Advon.NumberOfBuy (contracts/advon.sol#583) is not in mixedCase
Variable Advon._totalSupply (contracts/advon.sol#592) is not in mixedCase
Variable Advon._name (contracts/advon.sol#594) is not in mixedCase
Variable Advon._symbol (contracts/advon.sol#595) is not in mixedCase
Variable Advon._owner (contracts/advon.sol#599) is not in mixedCase
Variable Advon.BuyTax (contracts/advon.sol#615) is not in mixedCase
Variable Advon.SellTax (contracts/advon.sol#616) is not in mixedCase
Constant Advon.monthTime (contracts/advon.sol#619) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Advon.yearTime (contracts/advon.sol#621) is not in UPPER_CASE_WITH_UNDERSCORES
Variable Advon.FoundermonthsClaimed (contracts/advon.sol#627) is not in mixedCase
Variable Advon.FounderlastClaimedTime (contracts/advon.sol#628) is not in mixedCase
Variable Advon.FounderRemainingclaimableTokens (contracts/advon.sol#629) is not in mixedCase
Variable Advon.FoundersTotalClaimed (contracts/advon.sol#630) is not in mixedCase
Variable Advon.MarketingmonthsClaimed (contracts/advon.sol#632) is not in mixedCase
Variable Advon.MarketinglastClaimedTime (contracts/advon.sol#633) is not in mixedCase
Variable Advon.MarketingRemainingclaimableTokens (contracts/advon.sol#634) is not in mixedCase
Variable Advon.MarketingTotalClaimed (contracts/advon.sol#635) is not in mixedCase
Variable Advon.FoundersAllocation (contracts/advon.sol#649) is not in mixedCase
Variable Advon.MarketingAllocation (contracts/advon.sol#652) is not in mixedCase
Variable Advon.NumberOfResellers (contracts/advon.sol#663) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

```
Variable Advon.claim().claimablemonths_scope_1 (contracts/advon.sol#768) is too similar to Advon.claim().claimablemonths_scope_3 (contracts/advon.
81)
Variable Advon.claim().claimablemonths_scope_1 (contracts/advon.sol#768) is too similar to Advon.claim().claimablemonths_scope_5 (contracts/advon.
94)
Variable Advon.claim().claimablemonths_scope_3 (contracts/advon.sol#781) is too similar to Advon.claim().claimablemonths_scope_5 (contracts/advon.
94)
Variable Advon.transferFrom(address,address,uint256).locked_scope_0 (contracts/advon.sol#1149) is too similar to Advon.batchTransfer(address[],uin
]).locked_scope_3 (contracts/advon.sol#1055)
Variable Advon.transfer(address,uint256).locked_scope_0 (contracts/advon.sol#991) is too similar to Advon.batchTransfer(address[],uint256[]).locke
pe_3 (contracts/advon.sol#1055)
Variable Advon.privateSale1 (contracts/advon.sol#602) is too similar to Advon.privateSale2 (contracts/advon.sol#603)
Variable Advon.privateSale1 (contracts/advon.sol#602) is too similar to Advon.privateSale3 (contracts/advon.sol#604)
Variable Advon.privateSale2 (contracts/advon.sol#603) is too similar to Advon.privateSale3 (contracts/advon.sol#604)
Variable Advon.transfer(address,uint256).tradeable_scope_1 (contracts/advon.sol#992) is too similar to Advon.batchTransfer(address[],uint256[]).tr
le_scope_4 (contracts/advon.sol#1056)
Variable Advon.transferFrom(address,address,uint256).tradeable_scope_1 (contracts/advon.sol#1150) is too similar to Advon.batchTransfer(address[],
56[]).tradeable_scope_4 (contracts/advon.sol#1056)
Variable Advon.claim().transferrableTokens_scope_2 (contracts/advon.sol#769) is too similar to Advon.claim().transferrableTokens_scope_4 (contract
on.sol#782)
Variable Advon.claim().transferrableTokens_scope_2 (contracts/advon.sol#769) is too similar to Advon.claim().transferrableTokens_scope_6 (contract
on.sol#795)
Variable Advon.claim().transferrableTokens_scope_4 (contracts/advon.sol#782) is too similar to Advon.claim().transferrableTokens_scope_6 (contract
on.sol#795)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar

Advon.claim() (contracts/advon.sol#732-805) uses literals with too many digits:
        - require(bool)(transferrableTokens <= 8000000 * 10 ** decimals()) (contracts/advon.sol#748)
Advon.claim() (contracts/advon.sol#732-805) uses literals with too many digits:
        - require(bool)(transferrableTokens_scope_2 <= 5000000 * 10 ** decimals()) (contracts/advon.sol#771)
Advon.claim() (contracts/advon.sol#732-805) uses literals with too many digits:
        - require(bool)(transferrableTokens_scope_4 <= 5000000 * 10 ** decimals()) (contracts/advon.sol#784)
Advon.claim() (contracts/advon.sol#732-805) uses literals with too many digits:
        - require(bool)(transferrableTokens_scope_6 <= 5000000 * 10 ** decimals()) (contracts/advon.sol#797)
Advon.distribute() (contracts/advon.sol#808-823) uses literals with too many digits:
        - _mint(seedRound,5000000 * 10 ** decimals()) (contracts/advon.sol#811)
Advon.distribute() (contracts/advon.sol#808-823) uses literals with too many digits:
        - _mint(privateSale1,5000000 * 10 ** decimals()) (contracts/advon.sol#812)
Advon.distribute() (contracts/advon.sol#808-823) uses literals with too many digits:
        - _mint(privateSale2,5000000 * 10 ** decimals()) (contracts/advon.sol#813)
Advon.distribute() (contracts/advon.sol#808-823) uses literals with too many digits:
        - _mint(privateSale3,5000000 * 10 ** decimals()) (contracts/advon.sol#814)
Advon.distribute() (contracts/advon.sol#808-823) uses literals with too many digits:
        - _mint(liquidity,20000000 * 10 ** decimals()) (contracts/advon.sol#817)
Advon.distribute() (contracts/advon.sol#808-823) uses literals with too many digits:
        - _mint(reserve,10000000 * 10 ** decimals()) (contracts/advon.sol#818)
Advon.distribute() (contracts/advon.sol#808-823) uses literals with too many digits:
        - _mint(address(this),50000000 * 10 ** decimals()) (contracts/advon.sol#820)
Advon.Burn(address,uint256) (contracts/advon.sol#1186-1208) uses literals with too many digits:
        - require(bool)((burntTokens + _value) <= 5000000) (contracts/advon.sol#1191)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

Advon.__gap (contracts/advon.sol#1422) is never used in Advon (contracts/advon.sol#576-1423)
Advon.BuyTax (contracts/advon.sol#615) is never used in Advon (contracts/advon.sol#576-1423)
Advon.SellTax (contracts/advon.sol#616) is never used in Advon (contracts/advon.sol#576-1423)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable

Advon.BuyTax (contracts/advon.sol#615) should be constant
Advon.SellTax (contracts/advon.sol#616) should be constant
Advon.ecosystemFund (contracts/advon.sol#606) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant

initialize() should be declared external:
        - Advon.initialize() (contracts/advon.sol#681-685)
name() should be declared external:
        - Advon.name() (contracts/advon.sol#877-879)
Getowner() should be declared external:
        - Advon.Getowner() (contracts/advon.sol#881-883)
symbol() should be declared external:
        - Advon.symbol() (contracts/advon.sol#889-891)
totalSupply() should be declared external:
        - Advon.totalSupply() (contracts/advon.sol#913-915)
balanceOf(address) should be declared external:
        - Advon.balanceOf(address) (contracts/advon.sol#920-922)
transfer(address,uint256) should be declared external:
        - Advon.transfer(address,uint256) (contracts/advon.sol#969-1001)
approve(address,uint256) should be declared external:
        - Advon.approve(address,uint256) (contracts/advon.sol#1106-1110)
transferFrom(address,address,uint256) should be declared external:
        - Advon.transferFrom(address,address,uint256) (contracts/advon.sol#1128-1160)
increaseAllowance(address,uint256) should be declared external:
        - Advon.increaseAllowance(address,uint256) (contracts/advon.sol#1173-1177)
renounceOwnership() should be declared external:
        - Advon.renounceOwnership() (contracts/advon.sol#1217-1220)
decreaseAllowance(address,uint256) should be declared external:
        - Advon.decreaseAllowance(address,uint256) (contracts/advon.sol#1236-1245)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

Closing Summary

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

Some issues of High, Medium, and Informational severity were found. Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the Advon Platform. This audit does not provide a security or correctness guarantee of 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 multistep process. One audit cannot be considered enough. We recommend that the Advon Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. 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.

700+
Audits Completed

$16B
Secured

700K
Lines of Code Audited

Follow Our Journey

Audit Report
April, 2023

For

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com