

ANGELBLOCK

SMART CONTRACT AUDIT

zokyo

November 3rd, 2022 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.



TECHNICAL SUMMARY

This document outlines the overall security of the AngelBlock smart contracts, evaluated by Zokyo's Blockchain Security team.

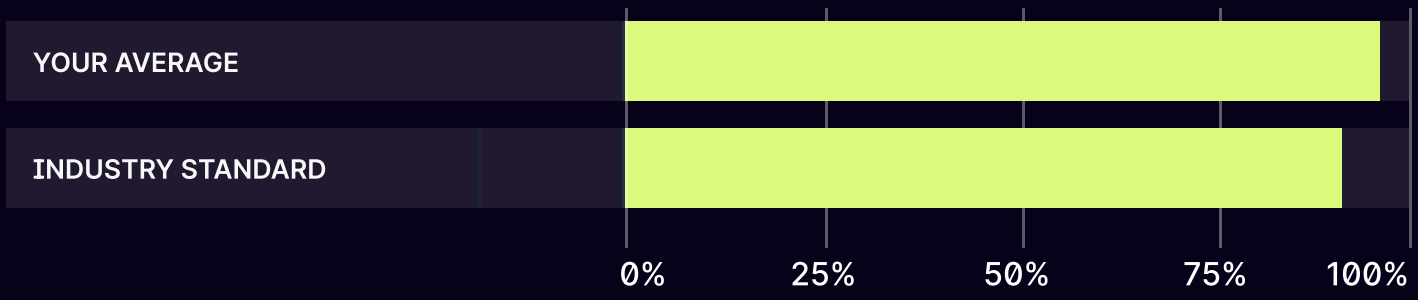
The scope of this audit was to analyze and document the AngelBlock smart contract codebase for quality, security, and correctness.

Contract Status



There were 2 critical issues found during the audit. (See Complete Analysis)

Testable Code



The testable code is 100%, which is above the industry standard of 95%. (See Complete Analysis)

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we at Zokyo recommend that the AngelBlock team put in place a bug bounty program to encourage further and active analysis of the smart contract.

Table of Contents

Auditing Strategy and Techniques Applied 3

Executive Summary 4

Structure and Organization of Document 5

Complete Analysis 6

Code Coverage and Test Results for all files written by the AngelBlock team 12

Code Coverage and Test Results for all files written by the Zokyo Security team 14

AUDITING STRATEGY AND TECHNIQUES APPLIED

The Smart contract's source code was taken from the AngelBlock repository.

Repository: https://github.com/angel-block/angelblock-contracts

Last commit: dd09eb9c99815cb906308acb8dbd163c70d7f280

Within the scope of this audit, Zokyo auditors have reviewed the following contract(s):

- presale/CommunityRound.sol
- libraries/LibSignature.sol
- utils/Configurable.sol

Throughout the review process, Zokyo Security ensures that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of resources, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of AngelBlock smart contracts. To do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. Part of this work includes writing a unit test suite using the Truffle testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01	Due diligence in assessing the overall code quality of the codebase.	02	Cross-comparison with other, similar smart contracts by industry leaders.
03	Testing contract logic against common and uncommon attack vectors.	04	Thorough manual review of the codebase, line by line.

Executive Summary

There were 2 critical issues found during the audit, alongside some of medium severity . All the mentioned findings have an effect only in case of specific conditions performed by the contract owner and the investors interacting with it. They are described in detail in the “Complete Analysis” section.

Contracts are well written and structured. Some findings during the audit shall have an impact on contract performance and security, so it is not fully production-ready, but minor editions by the development team shall make it ready for production.

STRUCTURE AND ORGANIZATION OF DOCUMENT

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the client team and the client team are aware of it, but they have chosen to not solved it. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that it can lead to a significant loss, funds may be lost or allocated incorrectly.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

SYSTEM OVERVIEW

The main use case of the contracts audited by the Zokyo team is to serve as components for the usdt staking scheme provided by AngelBlock. During the manual and testing stages of the contracts audit, multiple security issues were found. All those can be found in the following sections. Beside these findings, there are also remarks that have to be made about the overall security of the contracts submitted for audit.

During the initial assessment of the protocol, it has been discovered that the module manager component can perform some owner related actions that might help him gain funds. These include sending funds from the CommunityRound contract to arbitrary addresses. After further inspection, the Zokyo team established that these actions constitute a centralization risk for the security of the project's components. It's important to note that while some of these do not have a direct security impact on the contracts and project as a whole, there could be a scenario where this design can cause issues; such as complete loss of funds, undefined behavior due to either misconfiguration or malicious intent. Zokyo advises the team to acknowledge these design decisions and take extra care while operating the contracts in their current design.

After going deeper into the contracts, it is found that staking method (i.e. reserve) is exposed to replay attacks. Being exposed to replay attacks is a severe vulnerability in its own right. It adds another dimension to the severity though, as doing the replay attack messes up the balances of the investor since the balances are being overridden rather than adding up.

COMPLETE ANALYSIS

FINDINGS SUMMARY

#	Title	Risk
1	Exposure to replay attack	Critical
2	Balances messed up	Critical
3	USDT transfer is not safe	Medium
4	Centralization risk	Medium
5	Solidity version	Low
6	Unnecessary gas consumption	Informational
7	Decimals are not validated	Informational
8	Address argument not validated	Informational

COMPLETE ANALYSIS

CRITICAL | RESOLVED

Exposure to replay attack

CommunityRound.sol, method: `reserve(_request, _message, v, r, s), _request` along with `_message` and signature can be replayed. It is assumed that this is not intended since `balances[sender_] = _request.amount;` which implies that balances are set once, therefore this issue can be critical as it leads to funds being misallocated added to the fact that replay attacks need to be avoided anyway.

Recommendation:

Nonce uint mapping for senders.

CRITICAL | RESOLVED

Balances messed up

CommunityRound.sol, method: `reserve(_request, _message, v, r, s)`, balances of investors are being overwritten rather than adding up `balances[sender_] = _request.amount;`. Hence, funds of investors are misallocated. This issue is related to the replay attack issue and depending on the intention behind the method this issue might not be relevant if it is intended to have the investor call this method only once.

MEDIUM | RESOLVED

USDT transfer is not safe

CommunityRound.sol, method: `reserve(_request, _message, v, r, s)`, it is recommended as a best practice to use `SafeERC20` to transfer ERC20 tokens. Failed, unchecked ERC20 transfers might lead to reserving funds for an investor that is not actually deposited at the CommunityRound contract.

MEDIUM | RESOLVED

Centralization risk

Admin enjoys much authority granting roles in all the contracts which leads to enable him to manipulate funds. This can take place by calling `withdraw()` to take advantage of the asset by transferring all to a wallet of his choosing. Some methods can be more highly severe to be left out controlled by one wallet more than other methods. Recommendation Apply governance / use multisig wallets for certain roles.

LOW | RESOLVED

Solidity version

Lock the pragma to a specific version, since not all the EVM compiler versions support all the features, especially the latest ones which are kind of beta versions, So the intended behavior written in code might not be executed as expected. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, the latest compiler, which may have higher risks of undiscovered bugs.

Recommendation

fix version to `0.8.13`

INFORMATION | RESOLVED

Unnecessary gas consumption

CommunityRound.sol, method: `reserve(_request, _message, v, r, s)`, address of caller is being pushed into `investorsList` which is a private array of addresses, and it is not being read or used in any occasion. This storage serves no purpose, and it is considered an expensive operation.

INFORMATION | RESOLVED

Decimals are not validated

CommunityRound.sol, method: `configure(bytes)`, address of usdt_ better validate that it refers to ERC20 with decimals equal to 6 since variables `MAX_USDT_TO_RESERVE & MIN_USDT_TO_RESERVE` presume that this is the decimal value of the asset to be deposited.

INFORMATION | RESOLVED

Address argument not validated

CommunityRound.sol, method: `configure(bytes)` , the decoded variable `usdt_` is not verified to be non-zero.

Recommendation:

require statement on `usdt_` to be non-zero.

	CommunityRound	Configurable	LibSignature
Re-entrancy	Pass	Pass	Pass
Access Management Hierarchy	Pass	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass	Pass
Unexpected Ether	Pass	Pass	Pass
Delegatecall	Pass	Pass	Pass
Default Public Visibility	Pass	Pass	Pass
Hidden Malicious Code	Pass	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass	Pass
External Contract Referencing	Pass	Pass	Pass
Short Address/Parameter Attack	Pass	Pass	Pass
Unchecked CALL Return Values	Pass	Pass	Pass
Race Conditions/Front Running	Pass	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass	Pass
Floating Points and Precision	Pass	Pass	Pass
Tx.Origin Authentication	Pass	Pass	Pass
Signatures Replay	Pass	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by AngelBlock team

As a part of our work assisting the AngelBlock team in verifying the correctness of their contract code, we have checked the full set of unit tests prepared by the AngelBlock team.

It needs to be mentioned that the original code has a significant original coverage with testing scenarios provided by the AngelBlock team. All of them were also carefully checked by the team of auditors.

```
Alloc and TWAP
    ✓ Check TWAP - 1 USD unlock
Fundraising - check if voting added later works
    ✓ Should create new raise and in later added voting facet should be able to vote (4002ms)
Fundraising - collect vesting
    ✓ Should collect vesting for milestone (6840ms)
Fundraising - create raise
    ✓ Should create new raise (1264ms)
Fundraising - invest
    ✓ Should make new investment (invested: 2k USD, limit: 4k USD) (2117ms)
    ✓ Should make new investment (invested: 2k USD, limit: 2k USD) (2310ms)
    ✓ Should fail to make new investment (invested: 2k USD, limit: 1k USD) (1299ms)
    ✓ Should fail to make new investment at the wrong time (fundraising not started yet) (1346ms)
    ✓ Should fail to make new investment at the wrong time (fundraising finished) (1285ms)
    ✓ Should fail to make new investment (invested: 2k USD, hardcap: 1k USD) (1343ms)
Fundraising - postpone milestone
    ✓ Should be able to postpone (5331ms)
    ✓ Should not be able to postpone if voting already started (5498ms)
Fundraising - refund
    ✓ Should collect refund when raise failed (2421ms)
    ✓ Should fail to collect refund when raise succeed (2499ms)
    ✓ Should fail to collect refund before raise is finished (fundraising not started yet) (1207ms)
    ✓ Should fail to collect refund before raise is finished (fundraising ongoing) (1281ms)
Fundraising - voting
    ✓ Should work (3247ms)
    ✓ Should vote for milestone 3 times (13780ms)
    ✓ Should vote for milestone 5 times (15633ms)
    ✓ Should vote for milestone 9 times (18277ms)
    ✓ Should fail vote on failed raise (2201ms)
```

```
Alloc and Airdrop
    ✓ Check availability
    ✓ Forward (96ms)
    ✓ Upgrade and claim (306ms)
Alloc and CommunityRound
    ✓ Check availability
    ✓ Claim (124ms)
    ✓ Safeguard (129ms)
Alloc
    ✓ Check availability
    ✓ Forward (126ms)
    ✓ Claim (101ms)
    ✓ Safeguard (144ms)
EquityBadge
    ✓ Should mint new equity badge (216ms)
    ✓ Should not transfer badge (210ms)
    ✓ Should transfer badge (441ms)
CommunityRound
    ✓ Should test reserve (invested: 1200 USDT) (443ms)
    ✓ Should test reserve (invested: 1000 USDT) (421ms)
    ✓ Should test reserve (invested: 999 USDT) (121ms)
    ✓ Should test reserve (invested: 2000 USDT) (429ms)
    ✓ Should test reserve (invested: 2001 USDT) (402ms)
    ✓ Should fail reserve without allowance (87ms)
    ✓ Should fail reserve after deadline (64ms)
    ✓ Should test withdrawal (596ms)
TholosToken
    ✓ Should deploy token (1187ms)

43 passing (4m)
```

The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	% Uncovered Lines
libraries/LibSignature.sol	100	50	100	83.33	57
preseal/CommmunityRound.sol	97.67	62.5	87.5	86.27	... 246,306,318
utils/Configurable.sol	100	50	100	66.67	42
All files	**97.96**	**60**	**90.91**	**85**	

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security team

As part of our work assisting AngelBlock in verifying the correctness of their contract code, our team was responsible for writing integration tests using the Truffle testing framework.

Tests were based on the functionality of the code, as well as a review of the AngelBlock contract requirements for details about issuance amounts and how the system handles these.

CommunityRound

CommunityRound - Configure

- ✓ configure - revert because it is not called by admin (157ms)
- ✓ configure - revert because it is already configured

CommunityRound - reserve and withdraw

- ✓ reserve - check balance updated as expected (97ms)
- ✓ withdraw - verify proper withdrawal taking place (104ms)
- ✓ withdraw - revert since contract balance is zero
- ✓ reserve - revert since signer is not the correct sender (41ms)
- ✓ reserve - on edge of revert as expiry period is almost over (45ms)
- ✓ reserve - revert since expiry period is over
- ✓ reserve - revert since deadline period is over (40ms)
- ✓ reserve - revert since nonce is being replayed (52ms)
- ✓ reserve - revert since investor should only reserve once (56ms)
- ✓ reserve - revert since signer is not the privileged trustee
- ✓ reserve - revert since amount is less than minimum
- ✓ reserve - revert since amount is more than maximum
- ✓ reserve - revert since amount is more than maximum (53ms)
- ✓ reserve - revert since message does not match request
- ✓ reserve - revert because it is not yet configured (74ms)
- ✓ withdraw- revert because it is not yet configured (94ms)
- ✓ withdraw - revert because sender is not privileged to withdraw (82ms)

19 passing (10s)

The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	% Uncovered Lines
libraries/LibSignature.sol	100	100	100	100	
preseal/CommmunityRound.sol	100	100	100	100	
utils/Configurable.sol	100	100	100	100	
All files	**100**	**100**	**100**	**100**	

We are grateful to have been given the opportunity to work with the AngelBlock team.

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

Zokyo's Security Team recommends that the AngelBlock team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.



zokyo