Avatar

SMART CONTRACT AUDIT REPORT

Prepared by:

BlockAudit

Date Of Enrollment:

February 2nd, 2023 - February 5th, 2023

Visit : www.blockaudit.report

TABLE OF CONTENTS

SUMMARY

This Audit Report mainly focuses on the extensive security of **Avatar** Smart Contracts. With this report, we attempt to ensure the reliability and correctness of the smart contract by complete and rigorous assessment of the system's architecture and the smart contract codebase.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts producedby industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified inpublic;
- Provide more transparency on privileged activities once the protocol is live.

OVERVIEW

Project Summary

Project Name	Avatar
Language	Solidity
Platform	Polygon
Contract Address	0xC7728354f9fe0e43514B1227162D5B0E40FaD410

File Summary

ID	File Name
BKTB	BKTO-Bucket.sol
BKTA	BKTO-Avatar.sol
BKTP	BKTO-PaymentSplitter.sol

Date of Delivery	5 Feb 2023
Audit Methodology	Code Analysis. Automatic Assesment, Manual Review
Audit Result	**Passed** ✓✓
Audit Team	BlockAudit Report Team

FINDINGS

Critical	0	0.0%
High	0	0.0%
Medium	0	0.0%
Low	4	100.0%
Informational	0	0.0%
Ownership	0	0.0%

Vulnerability Findings Summary

ID	Type	Instances	Severity	Status
BKTP01	Use Custom Errors Rather Than Revert() / Require() Strings To Save Gas	55	Low	**Acknowledged**
BKTP02	Unlocked pragma used in contract	1	Low	Resolved
BKTP03	Unnecessary SLOADs and MLOADs in for-each loops	6	Low	Resolved
BKTP04	TODOs	1	Low	Resolved

BKTP01

Type	Use custom errors rather than revert() / require() strings to save gas
Severity	Low
File	Paymentsplitter.sol, Avatar.sol
Instances	55
Status	**Acknowledged**

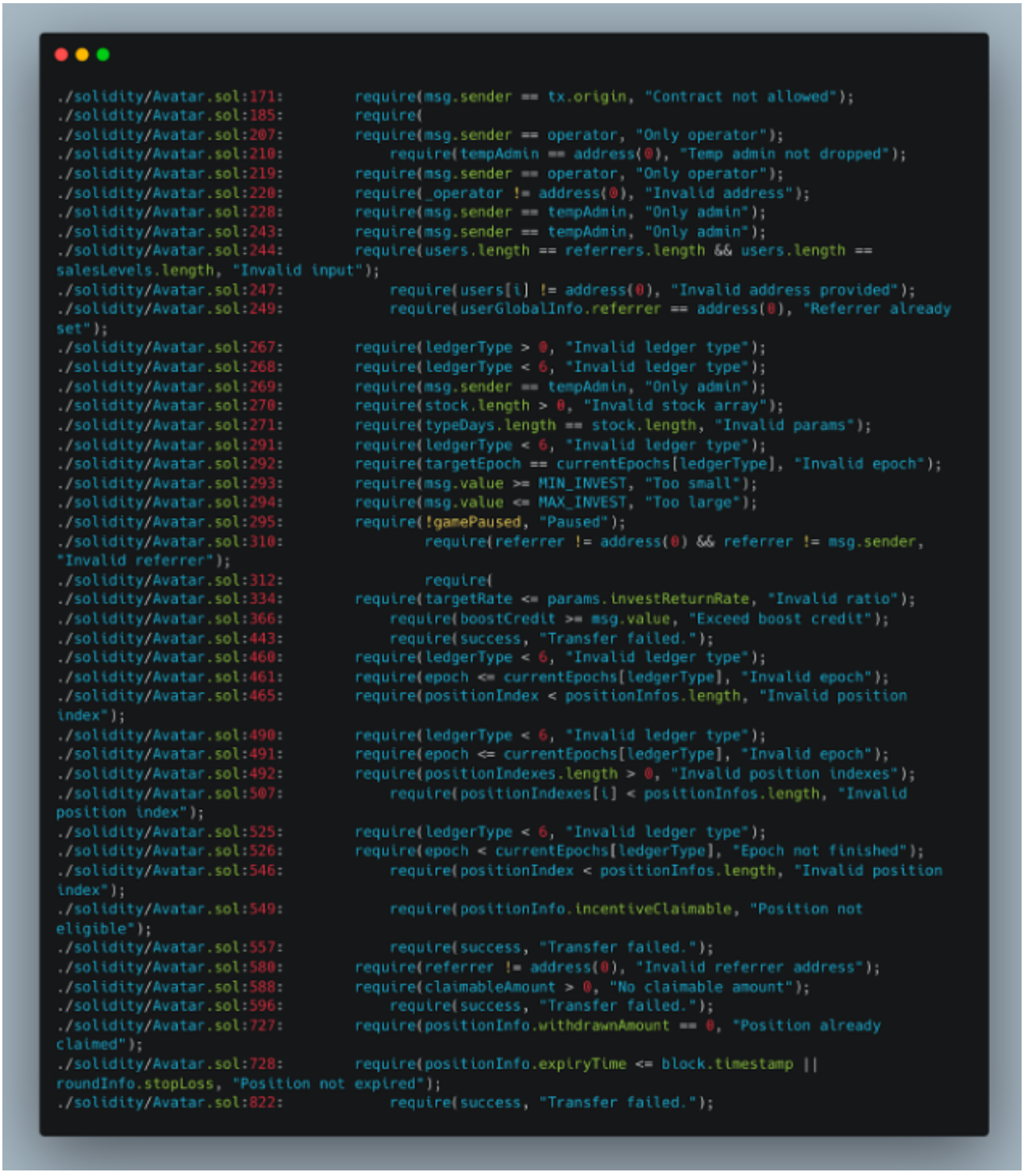
Description

Custom errors are available from solidity version 0.8.4. custom errors save ~50 gas each time they're hit by avoiding having to allocate and store the revert string. Not defining the strings also save deployment gas

Snapshot

```
./solidity/PaymentSplitter.sol:56:        require(msg.sender == tempAdmin, "PaymentSplitter: only temp
admin");
./solidity/PaymentSplitter.sol:68:        require(msg.sender == tempAdmin, "PaymentSplitter: only temp
admin");
./solidity/PaymentSplitter.sol:69:        require(payees.length == shares_.length, "PaymentSplitter:
payees and shares length mismatch");
./solidity/PaymentSplitter.sol:70:        require(payees.length > 0, "PaymentSplitter: no payees");
./solidity/PaymentSplitter.sol:165:        require(_shares[account] > 0, "PaymentSplitter: account has
no shares");
./solidity/PaymentSplitter.sol:169:        require(payment != 0, "PaymentSplitter: account is not due
payment");
./solidity/PaymentSplitter.sol:188:        require(_shares[account] > 0, "PaymentSplitter: account has
no shares");
./solidity/PaymentSplitter.sol:192:        require(payment != 0, "PaymentSplitter: account is not due
payment");
./solidity/PaymentSplitter.sol:224:        require(account != address(0), "PaymentSplitter: account is
the zero address");
./solidity/PaymentSplitter.sol:225:        require(shares_ > 0, "PaymentSplitter: shares are 0");
./solidity/PaymentSplitter.sol:226:        require(_shares[account] == 0, "PaymentSplitter: account
already has shares");
```

Snapshot



```
./solidity/Avatar.sol:171:        require(msg.sender == tx.origin, "Contract not allowed");
./solidity/Avatar.sol:185:        require(
./solidity/Avatar.sol:207:        require(msg.sender == operator, "Only operator");
./solidity/Avatar.sol:210:            require(tempAdmin == address(0), "Temp admin not dropped");
./solidity/Avatar.sol:219:        require(msg.sender == operator, "Only operator");
./solidity/Avatar.sol:220:        require(_operator != address(0), "Invalid address");
./solidity/Avatar.sol:228:        require(msg.sender == tempAdmin, "Only admin");
./solidity/Avatar.sol:243:        require(msg.sender == tempAdmin, "Only admin");
./solidity/Avatar.sol:244:        require(users.length == referrers.length && users.length ==
salesLevels.length, "Invalid input");
./solidity/Avatar.sol:247:            require(users[i] != address(0), "Invalid address provided");
./solidity/Avatar.sol:249:            require(userGlobalInfo.referrer == address(0), "Referrer already
set");
./solidity/Avatar.sol:267:        require(ledgerType > 0, "Invalid ledger type");
./solidity/Avatar.sol:268:        require(ledgerType < 6, "Invalid ledger type");
./solidity/Avatar.sol:269:        require(msg.sender == tempAdmin, "Only admin");
./solidity/Avatar.sol:270:        require(stock.length > 0, "Invalid stock array");
./solidity/Avatar.sol:271:        require(typeDays.length == stock.length, "Invalid params");
./solidity/Avatar.sol:291:        require(ledgerType < 6, "Invalid ledger type");
./solidity/Avatar.sol:292:        require(targetEpoch == currentEpochs[ledgerType], "Invalid epoch");
./solidity/Avatar.sol:293:        require(msg.value >= MIN_INVEST, "Too small");
./solidity/Avatar.sol:294:        require(msg.value <= MAX_INVEST, "Too large");
./solidity/Avatar.sol:295:        require(!gamePaused, "Paused");
./solidity/Avatar.sol:310:                require(referrer != address(0) && referrer != msg.sender,
"Invalid referrer");
./solidity/Avatar.sol:312:                require(
./solidity/Avatar.sol:334:        require(targetRate <= params.investReturnRate, "Invalid ratio");
./solidity/Avatar.sol:366:            require(boostCredit >= msg.value, "Exceed boost credit");
./solidity/Avatar.sol:443:            require(success, "Transfer failed.");
./solidity/Avatar.sol:460:        require(ledgerType < 6, "Invalid ledger type");
./solidity/Avatar.sol:461:        require(epoch <= currentEpochs[ledgerType], "Invalid epoch");
./solidity/Avatar.sol:465:        require(positionIndex < positionInfos.length, "Invalid position
index");
./solidity/Avatar.sol:490:        require(ledgerType < 6, "Invalid ledger type");
./solidity/Avatar.sol:491:        require(epoch <= currentEpochs[ledgerType], "Invalid epoch");
./solidity/Avatar.sol:492:        require(positionIndexes.length > 0, "Invalid position indexes");
./solidity/Avatar.sol:507:            require(positionIndexes[i] < positionInfos.length, "Invalid
position index");
./solidity/Avatar.sol:525:        require(ledgerType < 6, "Invalid ledger type");
./solidity/Avatar.sol:526:        require(epoch < currentEpochs[ledgerType], "Epoch not finished");
./solidity/Avatar.sol:546:            require(positionIndex < positionInfos.length, "Invalid position
index");
./solidity/Avatar.sol:549:            require(positionInfo.incentiveClaimable, "Position not
eligible");
./solidity/Avatar.sol:557:            require(success, "Transfer failed.");
./solidity/Avatar.sol:580:        require(referrer != address(0), "Invalid referrer address");
./solidity/Avatar.sol:588:        require(claimableAmount > 0, "No claimable amount");
./solidity/Avatar.sol:596:            require(success, "Transfer failed.");
./solidity/Avatar.sol:727:        require(positionInfo.withdrawnAmount == 0, "Position already
claimed");
./solidity/Avatar.sol:728:        require(positionInfo.expiryTime <= block.timestamp ||
roundInfo.stopLoss, "Position not expired");
./solidity/Avatar.sol:822:            require(success, "Transfer failed.");
```

BKTP02

Type	Unlocked pragma used in contract
Severity	Low
File	PaymentSplitter.sol
Instances	1
Status	Resolved

Description

Most of the contracts use an unlocked pragma (e.g., pragma solidity ^0.8.0) which is not fixed to a specific Solidity version. Locking the pragma helps ensure that contracts do not accidentally get deployed using a different compiler version with which they have been tested the most. Please use grep -R pragma . to find the unlocked pragma statements in the codebase

Remediation

Recommend locking pragmas to a specific Solidity version. Consider the compiler bugs in the following lists and ensure the contracts are not affected by them. It is also recommended to use the latest version of Solidity when deploying contracts (see Solidity docs).

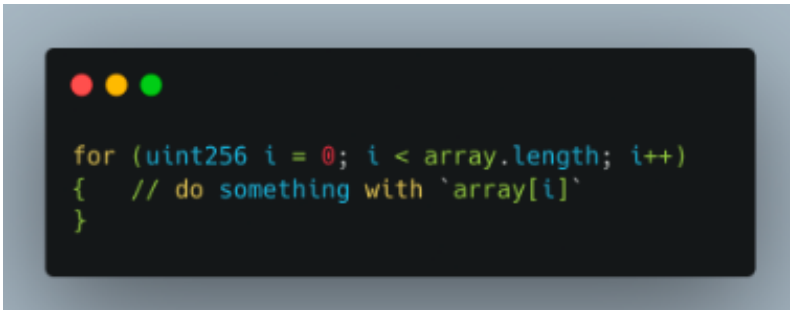
Snapshot

BKTP03

Type	Unnecessary SLOADs and MLOADs in for-each loops
Severity	Low
File	All files
Instances	6
Status	Resolved

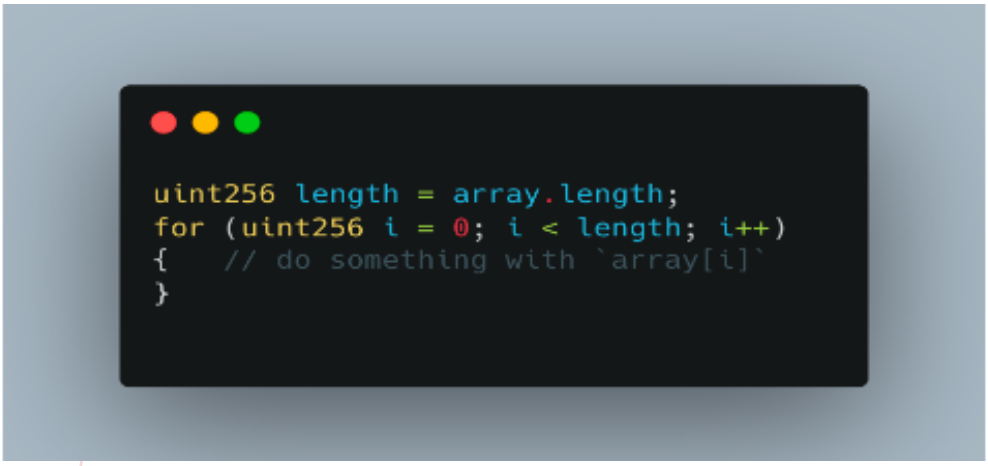
Description

There are many for loops that follows this for-each pattern:



In such for loops, the array.length is read on every iteration, instead of caching it once in a local variable and read it from there. Storage reads are much more expensive than reading local variables. Memory reads are a bit more expensive than reading local variables.

Remediation



Snapshot

```
./solidity/PaymentSplitter.sol:72:        for (uint256 i = 0; i < payees.length; i++) {
./solidity/Avatar.sol:246:        for (uint256 i = 0; i < users.length; ++i) {
./solidity/Avatar.sol:506:        for (uint256 i = 0; i < positionIndexes.length; ++i)
{/solidity/Avatar.sol:544:        for (uint256 i = 0; i < positionIndexes.length; ++i)
{/solidity/Avatar.sol:570:        for (uint256 i = 0; i < users.length; ++i) {
./solidity/Bucket.sol:27:        for (uint16 i = 0; i < stock.length; ++i) {
```

BKTP04

Type	TODOs
Severity	Low
File	Avatar.sol
Instances	1
Status	Resolved

Description

There are TODOs left in the code. While this does not cause any direct issue, it indicates a bad smell and uncertainty and makes it harder for the auditor to make assumptions on the codebase.

Remediation

Remove TODOs before Deployment

Snapshot

APPENDIX

Auditing Approach and Methodologies applied

The Block Audit Report team has performed rigorous testing of the project including the analysis of the code design patterns where we reviewed the smart contract architecture to ensure it is structured along with the safe use of standard inherited contracts and libraries. Our team also conducted a formal line by line inspection of the Smart Contract i.e., a manual review, to find potential issues including but not limited to

- Race conditions
- Zero race conditions approval attacks
- Re-entrancy
- Transaction-ordering dependence
- Timestamp dependence
- Check-effects-interaction pattern (optimistic accounting)
- Decentralized denial-of-service attacks
- Secure ether transfer pattern
- Guard check pattern
- Fail-safe mode
- Gas-limits and infinite loops
- Call Stack depth

In the Unit testing Phase, we coded/conducted custom unit tests written against each function in the contract to verify the claimed functionality from our client. In Automated Testing, we tested the Smart Contract with our standard set of multifunctional tools to identify vulnerabilities and security flaws. The code was tested in collaboration of our multiple team members and this included but not limited to;

- Testing the functionality of the Smart Contract to determine proper logic has been followed throughout the whole process.
- Analyzing the complexity of the code in depth and in detail line-by-line manual review of the code.
- Deploying the code on testnet using multiple clients to run live tests.
- Analyzing failure preparations to check how the Smart Contract performs in case of any bugs and vulnerabilities.
- Checking whether all the libraries used in the code are on the latest version.
- Analyzing the security of the on-chain data.

Issue Categories:

Every issue in this report was assigned a severity level from the following:

Critical Severity Issues

Issues of Critical Severity leaves smart contracts vulnerable to major exploits and can lead to asset loss and data loss. These can have significant impact on the functionality/performance of the smart contract.
We recommend these issues must be fixed before proceeding to MainNet..

High Severity Issues

Issues of High Severity are not as easy to exploit but they might endanger the execution of the smart contract and potentially create crucial problems.
Fixing these issues is highly recommended before proceeding to MainNet.

Medium Severity Issues

Issues on this level are not a major cause of vulnerability to the smart contract, they cannot lead to data-manipulations or asset loss but may affect funtionality.
It is important to fix these issues before proceeding to MainNet.

Low Severity Issues

Issues at this level are very low in their impact on the overall functionality and execution of the smart contract. These are mostly code-level violations or improper formatting.
These issues can be remain unfixed or can be fixed at a later date if the code is redeployed or forked.

Informational Findings

These are finding that our team comes accross when manually reviewing a smart contract which are important to know for the owners as well as users of a contract.
These issues must be acknowledged by the owners before we publish our report.

Ownership Priviledges

Owner of a smart contract can include certain rights and priviledges while deploying a smart contract that might be hidden deep inside the codebase and may make the project vulnerable to rug-pulls or other types of scams.
We at BlockAudit believe in transparency and hence we showcase Ownership priviledges separately so the owner as well as the investors can get a better understanding about the project.

DISCLAIMER

This is a limited report on our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for the client to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that the client should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for the client to conduct the client's own independent investigations before making any decisions. We go into more detail on this in the below disclaimer below – please make sure to read it in full.

By reading this report or any part of it, the client agrees to the terms of this disclaimer. If the client does not agree to the terms, then please immediately cease reading this report, and delete and destroy any/all copies of this report downloaded and/or printed by the client. This report is provided for information purposes only and stays on a non-reliance basis, and does not constitute investment advice. No one/ NONE shall have any rights to rely on the report or its contents, and BlockAudit and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives).

(BlockAudit) owes no duty of care towards the client or any other person, nor does BlockAudit claim any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and BlockAudit hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effects in relation to the report.

Except and only to the extent that it is prohibited by law, BlockAudit hereby excludes all liability and responsibility, and neither the client nor any other person shall have any claim against BlockAudit, for any amount or kind of loss or damage that may result to the client or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report.

The analysis of the security is purely based on the received smart contracts alone. No related/third-party smart contracts, applications or operations were reviewed for security. No product code has been reviewed.

Note: 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 **AVATAR** team put a bug bounty program in place to encourage further analysis of the smart contracts by other third parties

About BlockAudit

BlockAudit is an industry leading security organisation that helps web3 blockchain based projects with their security and correctness of their smart-contracts. With years of experience we have a dedicated team that is capable of performing audits in a wide variety of languages including HTML, PHP, JS, Node, React, Native, Solidity, Rust and other Web3 frameworks for DApps, DeFi, GameFi and Metaverse platforms.

With a mission to make web3 a safe and secure place BlockAudit is commited to provide it's partners with a budget and investor friendly security Audit Report that will increase the value of their projects significantly.

 *www.blockaudit.report*

 *team@blockaudit.report*

 *@BlockAudit*

 *github.com/Block-Audit-Report*