BlockÂpex

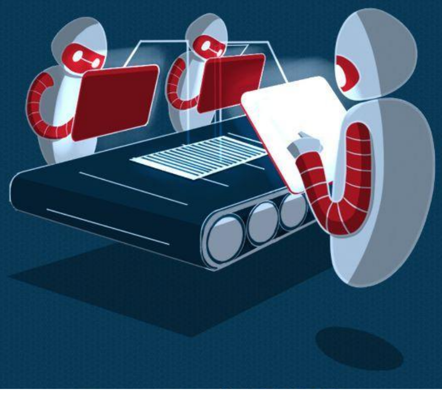
SMART CONTRACT SECURITY ANALYSIS REPORT

```
pragma solidity 0.7.0;
contract Contract {

    function hello() public returns (string) {
        return "Hello World!";
    }

    function findVulnerability() public returns (string) {
        return "Finding Vulnerability";
    }

    function solveVulnerability() public returns (string) {
        return "Solve Vulnerability";
    }
}
```



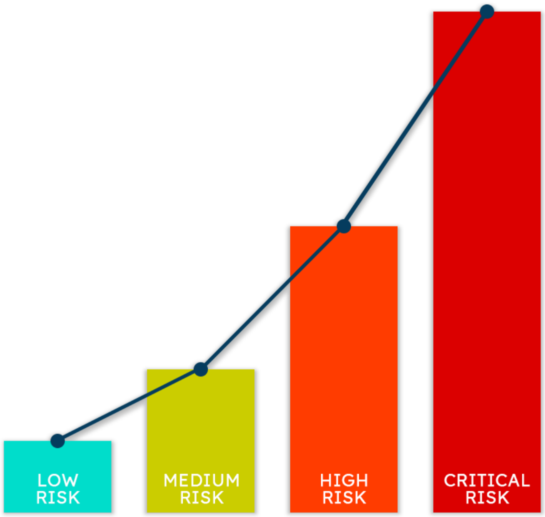
PREFACE

Objectives

The purpose of this document is to highlight the identified bugs/issues in the provided codebase. This audit has been conducted in a closed and secure environment, free from influence or bias of any sort. This document may contain confidential information about IT systems/architecture and intellectual property of the client. It also contains information about potential risks and the processes involved in mitigating/exploiting the risks mentioned below.

The usage of information provided in this report is limited, internally, to the client. However, this report can be disclosed publicly with the intention to aid our growing blockchain community; under the discretion of the client.

Key understandings



CRITICAL RISK xxxx	Critical vulnerabilities are too easy to exploit and can lead to damages/loss in assets or manipulations.
HIGH RISK xxx	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution.
MEDIUM RISK xx	Medium-level vulnerabilities are equally imperative to fix but they tend to have minimal impact on asset loss or data manipulations.
LOW RISK x	Lowest-level vulnerabilities, informational errors, violating code styles/practices usually can't affect smart contract execution; hence they can be ignored.

TABLE OF CONTENTS

INTRODUCTION

BlockApex (Auditor) was contracted by Chainpals (Client) for the purpose of conducting a Smart Contract Audit/Code Review. This document presents the findings of our analysis which took place on 15th June 2022.

Name
ChainpalsTransaction
Auditor
Kaif Ahmed | Mohammad Memon
Platform
Ethereum/Solidity/BCS
Type of review
Manual Code Review | Automated Code Review
Methods
Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
BSC Scan/Contract address
https://bscscan.com/address/0x491103d0a391c344c75e960f1fdff94b83c5b1a3#code
Documentation
https://chainpals.io/assets/document/ChainpalsLightpaper.pdf
Document log
Initial Audit: 16th June 2022
Final Audit: 22 June 2022

Scope

The git-repository shared was checked for common code violations along with vulnerability-specific probing to detect **major issues/vulnerabilities**. Some specific checks are as follows:

Code review		**Functional review**
Reentrancy	Unchecked external call	Business Logics Review
Ownership Takeover	ERC20 API violation	Functionality Checks
Timestamp Dependence	Unchecked math	Access Control & Authorization
Gas Limit and Loops	Unsafe type inference	Escrow manipulation
DoS with (Unexpected) Throw	Implicit visibility level	Token Supply manipulation
DoS with Block Gas Limit	Deployment Consistency	Asset’s integrity
Transaction-Ordering Dependence	Repository Consistency	User Balances manipulation
Style guide violation	Data Consistency	Kill-Switch Mechanism
Costly Loop		Operation Trails & Event Generation

Project Overview

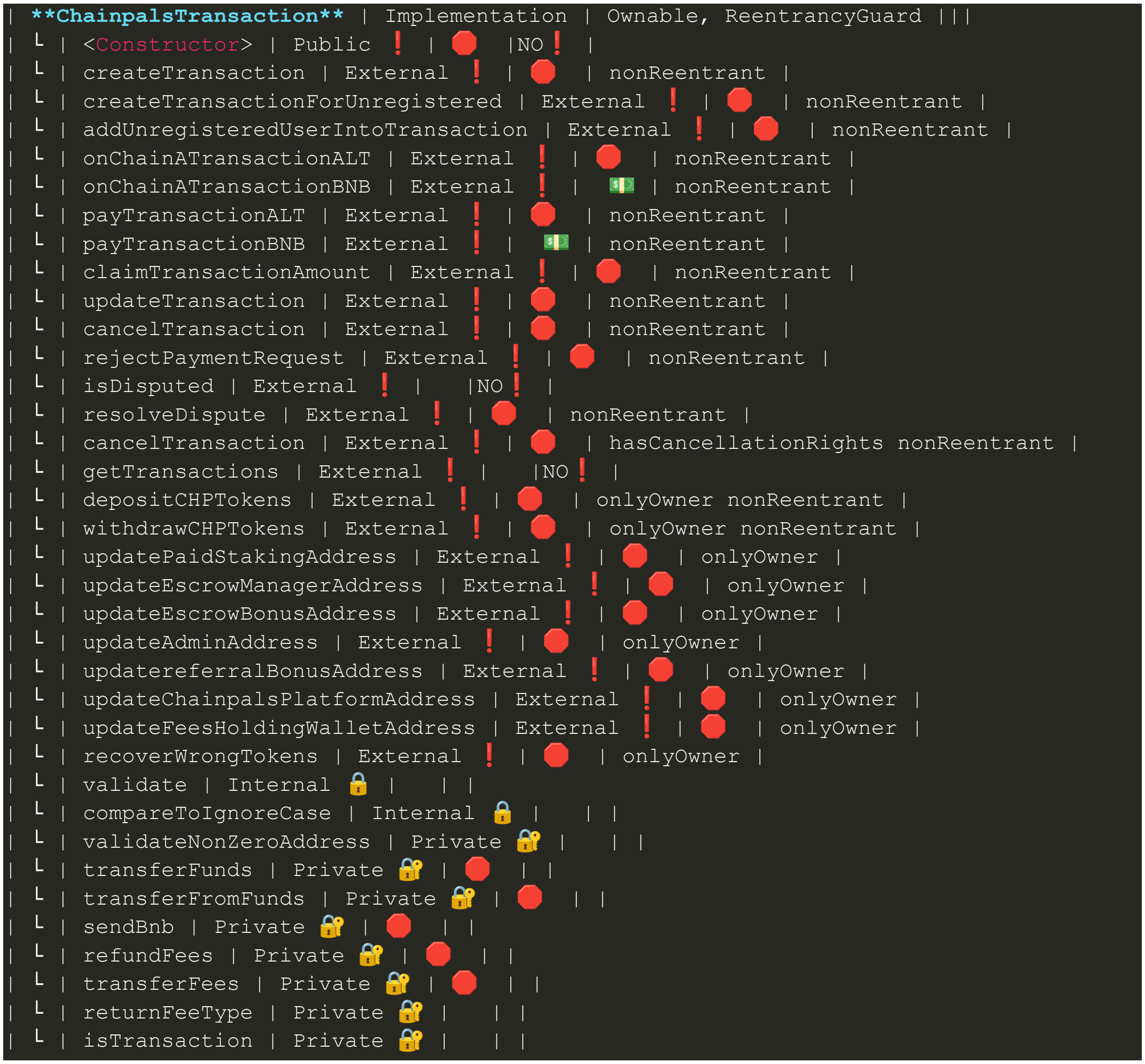
Chainpals transaction contract is responsible for handling the multi-phased transactions that take place between a buyer and a seller, each overlooked by escrow managers to make sure everything goes smoothly.

System Architecture

The trio of Chainpals contracts form a system which allows end users to meet, setup transaction details (allowing payments in any BEP20 token) while making sure that the transaction proceeds only if both the parties agree on the rules. The system also has their own BEP20 token called ChainpalsToken. The actors are incentivized to use these native tokens, which allows them to avail special discounts on fees. People are also encouraged to tell others about this protocol, for which they get bonuses in the form of NFTs and a share.

Methodology & Scope

The code came to us in the form of a zip, containing a truffle directory, the contract and the tests. Initially, we ran the contract and tested the functionality of all the functions manually. After that, we moved to Foundry to try all kinds of scenarios. After all the logical and functional testing, we moved to code optimizations and solidity design patterns to ensure consistency and readability.



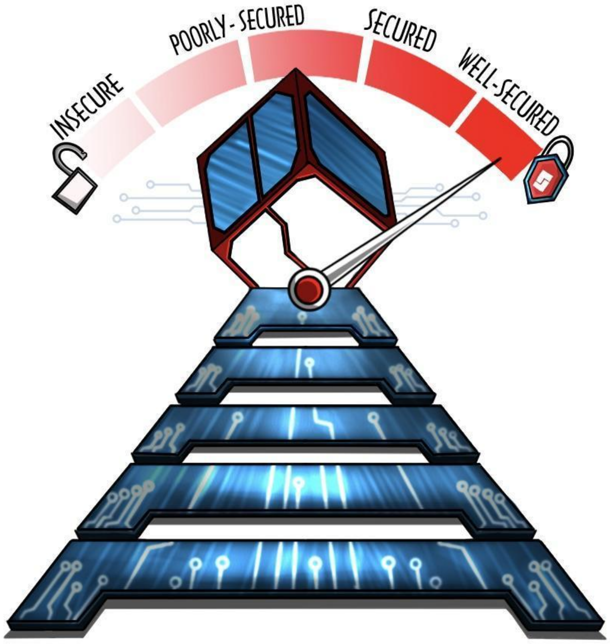
```
| **ChainpalsTransaction** | Implementation | Ownable, ReentrancyGuard |||
| └ | <Constructor> | Public ❗️ | 🛑  |NO❗️ |
| └ | createTransaction | External ❗️ | 🛑  | nonReentrant |
| └ | createTransactionForUnregistered | External ❗️ | 🛑  | nonReentrant |
| └ | addUnregisteredUserIntoTransaction | External ❗️ | 🛑  | nonReentrant |
| └ | onChainATransactionALT | External ❗️ | 🛑  | nonReentrant |
| └ | onChainATransactionBNB | External ❗️ |  💵 | nonReentrant |
| └ | payTransactionALT | External ❗️ | 🛑  | nonReentrant |
| └ | payTransactionBNB | External ❗️ |  💵 | nonReentrant |
| └ | claimTransactionAmount | External ❗️ | 🛑  | nonReentrant |
| └ | updateTransaction | External ❗️ | 🛑  | nonReentrant |
| └ | cancelTransaction | External ❗️ | 🛑  | nonReentrant |
| └ | rejectPaymentRequest | External ❗️ | 🛑  | nonReentrant |
| └ | isDisputed | External ❗️ |   |NO❗️ |
| └ | resolveDispute | External ❗️ | 🛑  | nonReentrant |
| └ | cancelTransaction | External ❗️ | 🛑  | hasCancellationRights nonReentrant |
| └ | getTransactions | External ❗️ |   |NO❗️ |
| └ | depositCHPTokens | External ❗️ | 🛑  | onlyOwner nonReentrant |
| └ | withdrawCHPTokens | External ❗️ | 🛑  | onlyOwner nonReentrant |
| └ | updatePaidStakingAddress | External ❗️ | 🛑  | onlyOwner |
| └ | updateEscrowManagerAddress | External ❗️ | 🛑  | onlyOwner |
| └ | updateEscrowBonusAddress | External ❗️ | 🛑  | onlyOwner |
| └ | updateAdminAddress | External ❗️ | 🛑  | onlyOwner |
| └ | updatereferralBonusAddress | External ❗️ | 🛑  | onlyOwner |
| └ | updateChainpalsPlatformAddress | External ❗️ | 🛑  | onlyOwner |
| └ | updateFeesHoldingWalletAddress | External ❗️ | 🛑  | onlyOwner |
| └ | recoverWrongTokens | External ❗️ | 🛑  | onlyOwner |
| └ | validate | Internal 🔒 |   | |
| └ | compareToIgnoreCase | Internal 🔒 |   | |
| └ | validateNonZeroAddress | Private 🔐 |   | |
| └ | transferFunds | Private 🔐 | 🛑  | |
| └ | transferFromFunds | Private 🔐 | 🛑  | |
| └ | sendBnb | Private 🔐 | 🛑  | |
| └ | refundFees | Private 🔐 | 🛑  | |
| └ | transferFees | Private 🔐 | 🛑  | |
| └ | returnFeeType | Private 🔐 |   | |
| └ | isTransaction | Private 🔐 |   | |
```

AUDIT REPORT

Executive Summary

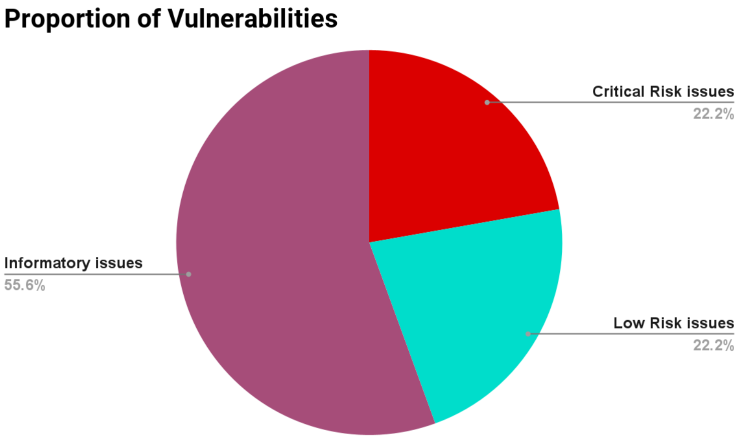
The analysis indicates that all of the functionalities in the contract audited are **working properly**.

Our team performed a technique called “Filtered Audit”, where the contract was separately audited by two individuals. After their thorough and rigorous process of manual testing, an automated review was carried out using Surya. All the flags raised were manually reviewed and re-tested.



Our team found:

# of issues	Severity of the risk
2	Critical Risk issue(s)
0	High Risk issue(s)
0	Medium Risk issue(s)
2	Low Risk issue(s)
5	Informatory issue(s)



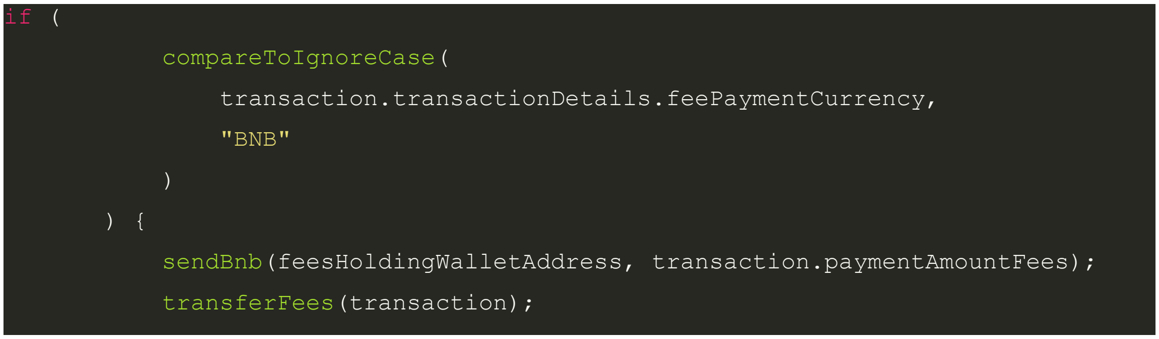
Findings

#	Findings	Risk	Status
1.	Missing core functionality	Critical	Acknowledged
2.	Misleading functionality	Critical	Acknowledged
3.	Missing zero address checks	Low	Fixed
4.	Unnecessary conversion	Low	Fixed
5.	Anti-pattern check	Informatory	Fixed
6.	Inconsistency in error messages	Informatory	Fixed
7.	Spelling mistakes	Informatory	Fixed
8.	Follow solidity design pattern	Informatory	Fixed
9.	Inconsistent code writing	Informatory	Fixed

Critical risk issues

1. Missing core functionality

Description: Contract seems to have a functionality of deducting fees in BNB. The claim transaction amount function has a check that the feePaymentCurrency variable is BNB but the contract is missing the functionality of collecting fees in BNB.



```
if (
            compareToIgnoreCase(
                transaction.transactionDetails.feePaymentCurrency,
                "BNB"
            )
        ) {
            sendBnb(feesHoldingWalletAddress, transaction.paymentAmountFees);
            transferFees(transaction);
```

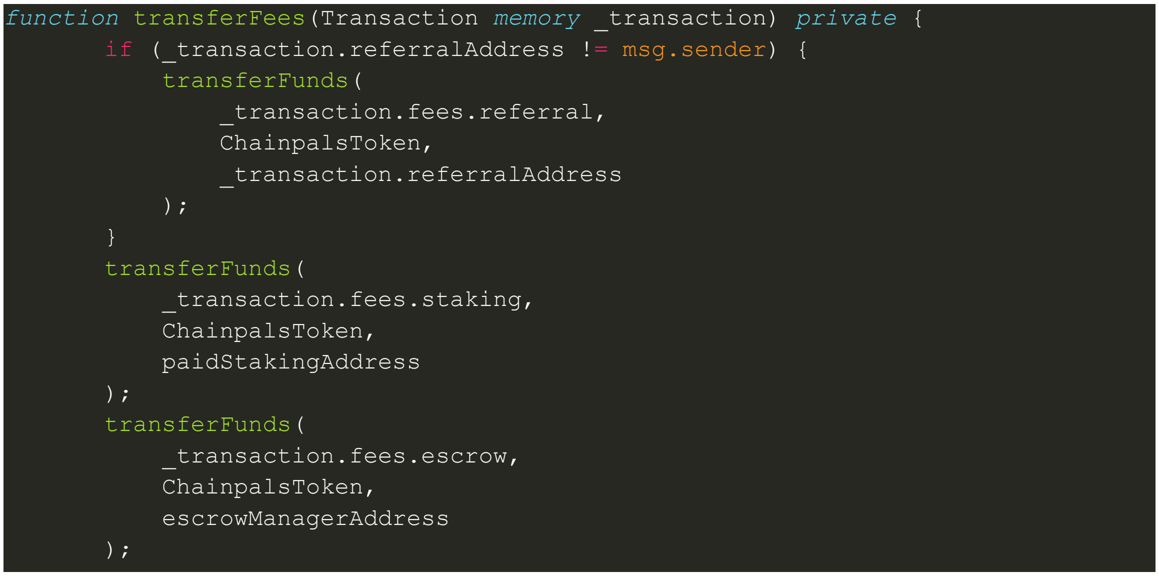
Remedy: Write a proper code to collect fees in BNB instead of CHP when the feePaymentCurrency is set to BNB.

Status: Acknowledged

Developer Response: If user has created transaction using two different currency platform Fees in BNB & Transaction payment in USDT then in smart contract there is one function name : “onChainATransactionBNB” using we are collecting platform fees(BNB) and if user has created transaction in same currency for payment and platform fees(BNB) then on the time of make payment (function name : payTransactionBNB) will collect both platform fee & payment in single transaction.

2. Misleading functionality

Description: The contract contains a function called *transferFees()* which calls the function *transferFunds()* to send fees to five different wallets. The *transferFunds()* has a hardcoded fee token address set to ChainpalsToken. Regardless of what token the user sets, the fee is always deducted in terms of the ChainpalsTokens. This is misleading because it does not function the way it is mentioned in the document.



```
function transferFees(Transaction memory _transaction) private {
       if (_transaction.referralAddress != msg.sender) {
           transferFunds(
               _transaction.fees.referral,
               ChainpalsToken,
               _transaction.referralAddress
           );
       }
       transferFunds(
           _transaction.fees.staking,
           ChainpalsToken,
           paidStakingAddress
       );
       transferFunds(
           _transaction.fees.escrow,
           ChainpalsToken,
           escrowManagerAddress
       );
```

Remedy: Write proper implementation to go forward with the method written in the documentation, or update the documentation to go along with the existing code.
Status: Acknowledged

Auditor's Response: Since the specs document was not clear and auditors made the wrong assumptions. It is necessary for the user to hold CHP tokens in order to create/claim transactions because protocol only supports CHP tokens for transactionFees. Also it is recommended to clear/mention this spec in public doc for users.

Low risk issues

3. Missing zero address checks

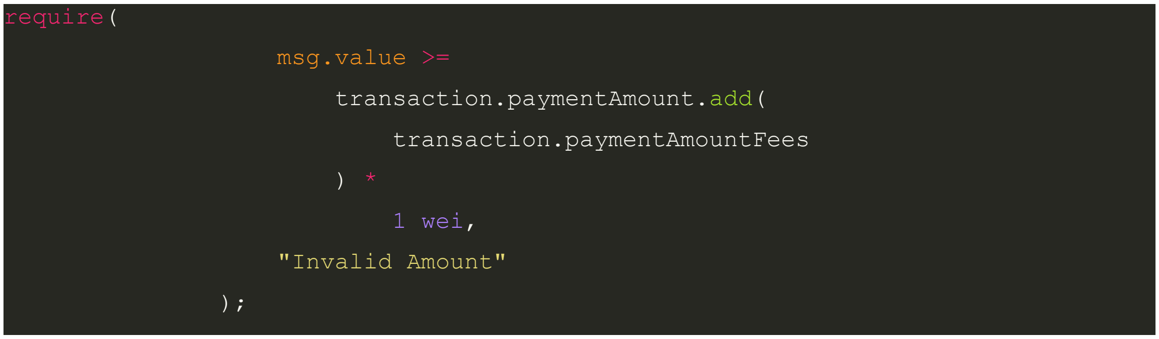
Description: The constructor accepts several address parameters, none of which are being checked for zero address. There is a *validateNonZeroAddress()* that checks for zero addresses, which can be used here. Here are some other functions missing zero address checks:

- *createTransaction()*
- *createTransactionForUnregistered()*
- *addUnregisteredUserIntoTransaction()*

Status: Fixed as per BlockApex recommendation.

4. Unnecessary conversion

Description: In the function *onChainATransactionBnb()*, there is a require statement which checks for *(paymentAmountFees * 1 wei <= msg.value)*. This operation is unnecessary. It is like multiplying the entire amount with 1, which is inconsequential.



```
require(
                   msg.value >=
                       transaction.paymentAmount.add(
                           transaction.paymentAmountFees
                       ) *
                           1 wei,
                   "Invalid Amount"
               );
```

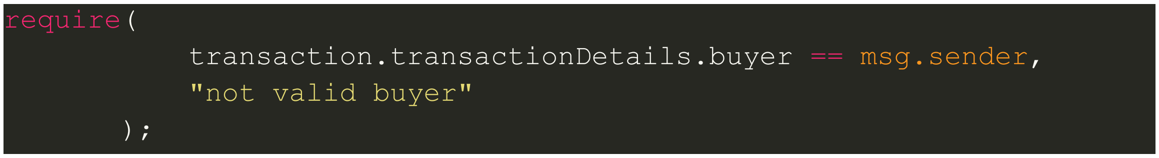
Remedy: Remove the unnecessary conversion.

Status: Fixed as per BlockApex recommendation.

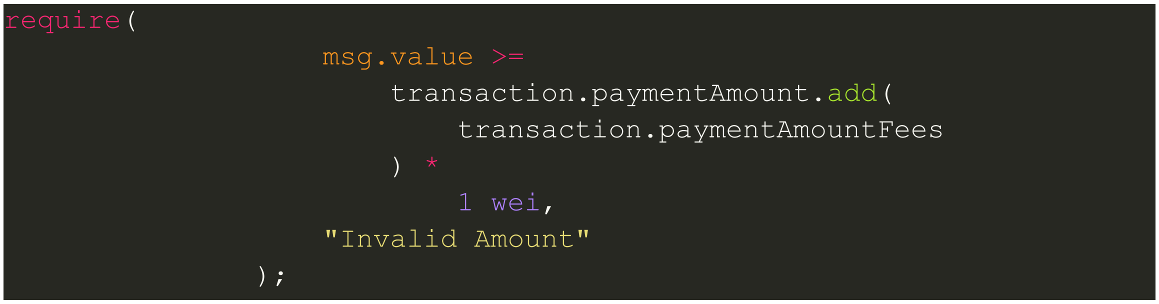
Informatory issues and Optimization

5. Anti-pattern check

Description: Conventionally, the global variable is on the left hand side of the comparison operator, with the local variable or the function parameter on the right hand side. Most checks in the code go against this. The code is not committed to one pattern, with the variables reversed in some cases.



```
require(
           transaction.transactionDetails.buyer == msg.sender,
           "not valid buyer"
       );
```

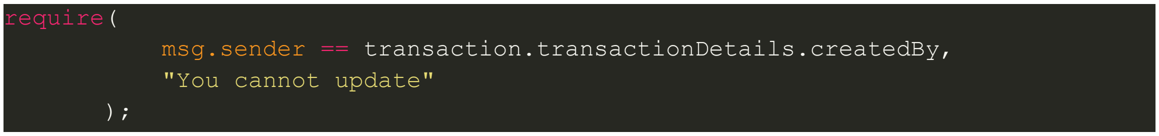


```
require(
                   msg.value >=
                       transaction.paymentAmount.add(
                           transaction.paymentAmountFees
                       ) *
                           1 wei,
                   "Invalid Amount"
               );
```

Status: Fixed as per BlockApex recommendation.

6. Inconsistency in error messages

Description: The error messages in the require checks are inconsistent. Even for the same check, each function has a different error message. Also, the error messages should be meaningful. At the moment, some messages in the code do not tell the user what the error is supposed to mean.



```
require(
           msg.sender == transaction.transactionDetails.createdBy,
           "You cannot update"
       );
```

Status: Fixed as per BlockApex recommendation.

7. Spelling mistakes

Description: There are several cases of spelling mistakes in the code.

```
address public referelBonusAddress;
```

```
enum PAYMENT {
        INSTANT,
        MIESTONE
    }
```

```
require(isTransaction(_uid) == false, "Invalid transction id");
```

Status: Fixed as per BlockApex recommendation.

8. Follow solidity design pattern

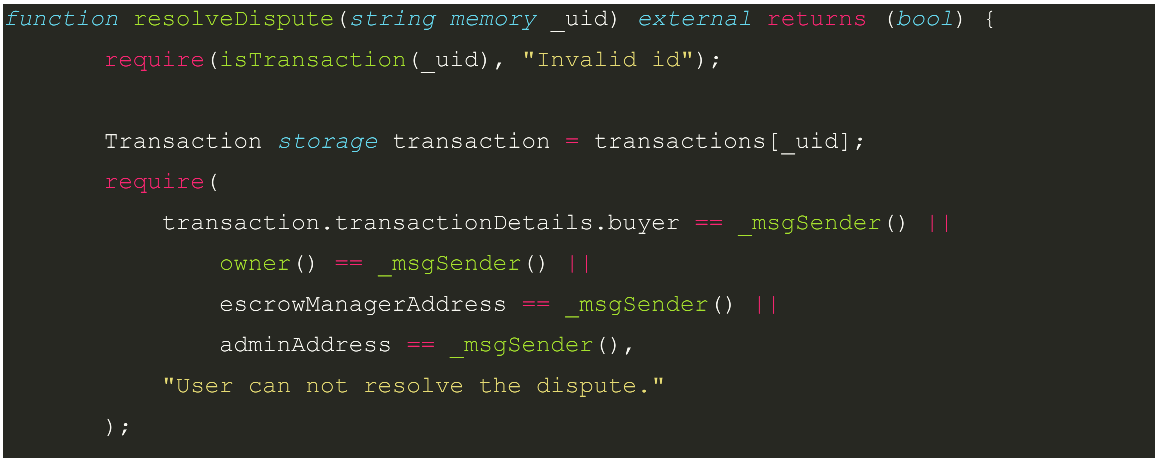
Description: As stated in the Solidity style guide, the functions should be grouped according to their visibility and ordered:

- constructor
- receive function (if exists)
- fallback function (if exists)
- external
- public
- internal
- Private

Status: Fixed as per BlockApex recommendation.

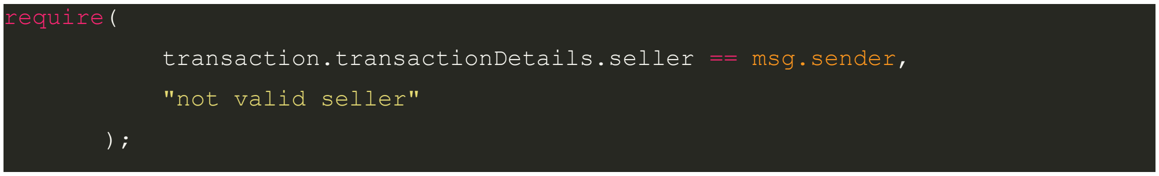
9. Inconsistent code writing

Description: The code has used both msg.sender and *msgSender()* from the Context library. It is suggested that you stick to one and use it throughout the code.



```
function resolveDispute(string memory _uid) external returns (bool) {
        require(isTransaction(_uid), "Invalid id");

        Transaction storage transaction = transactions[_uid];
        require(
            transaction.transactionDetails.buyer == _msgSender() ||
                owner() == _msgSender() ||
                escrowManagerAddress == _msgSender() ||
                adminAddress == _msgSender(),
            "User can not resolve the dispute."
        );
```



```
require(
            transaction.transactionDetails.seller == msg.sender,
            "not valid seller"
        );
```

Status: Fixed as per BlockApex recommendation.

DISCLAIMER

The smart contracts provided by the client for audit purposes have been thoroughly analyzed in compliance with the global best practices till date w.r.t cybersecurity vulnerabilities and issues in smart contract code, the details of which are enclosed in this report.

This report is not an endorsement or indictment of the project or team, and they do not in any way guarantee the security of the particular object in context. This report is not considered, and should not be interpreted as an influence, on the potential economics of the token, its sale or any other aspect of the project.

Crypto assets/tokens are results of the emerging blockchain technology in the domain of decentralized finance and they carry with them high levels of technical risk and uncertainty. No report provides any warranty or representation to any third-Party in any respect, including regarding the bug-free nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third-party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project.

Smart contracts are deployed and executed on a blockchain. The platform, its programming language, and other software related to the smart contract can have its vulnerabilities that can lead to hacks. The scope of our review is limited to a review of the Solidity code and only the Solidity code we note as being within the scope of our review within this report. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond Solidity that could present security risks.

This audit cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.