SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Paydece
Date: March 01, 2023

Document

Name	Smart Contract Code Review and Security Analysis Report for Paydece
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	Escrow
Platform	EVM
Language	Solidity
Methodology	Link
Website	http://paydece.io/
Changelog	17.01.2023 - Initial Review 03.02.2023 - Second Review 16.02.2023 - Third Review 01.03.2023 - Fourth Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Paydece (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project is smart contracts in the repository:

Initial review scope

Repository	https://github.com/PayDece/paydece-contracts
Commit	3f71651c954f1f644f943d5d4d53dd8c7ad351e4
Functional Requirements	Link
Technical Requirements	Link
Contracts	File: ./contracts/PaydeceEscrowV3.sol SHA3: 31d41e42f3e4a84b739bad2807ea1b500f108a7d61bef674eb80747bd069f656 File: ./contracts/USDTToken.sol SHA3: d39fae5484dd9d3855745c8146438266c79b01763a1fc63d3779ae2e0931a061

Second review scope

Repository	https://github.com/PayDece/paydece-contracts
Commit	fa22e1f0e8648640c386d56d9f4c33658a3bb861
Functional Requirements	Link
Technical Requirements	Link
Contracts	File: ./contracts/PaydeceEscrowV3.sol SHA3: 9405a0a0a917dddd72d959ae202e1470c73b7ff9fddb9c60de04bd482d2d1200 File: ./contracts/USDTToken.sol SHA3: d39fae5484dd9d3855745c8146438266c79b01763a1fc63d3779ae2e0931a061

Third review scope

Repository	https://github.com/PayDece/paydece-contracts
Commit	5127f7038c21221475651ae94c5c361384094f5c
Functional Requirements	Link

Technical Requirements	Link
Contracts	File: ./contracts/PaydeceEscrowV3.sol SHA3: 7b5c5d7df3ea3c592729d843cdbf99802ffb5558850b585e42a951c1b3e857f6 File: ./contracts/USDTToken.sol SHA3: acfa51095aa02b961837564b1bc4295015e2fd709747f3c6cb1f3f38014b9f2d

Fourth review scope

Repository	https://github.com/PayDece/paydece-contracts
Commit	3f6775db80cf7905dfa3443fe2ebf606e1a235f7
Functional Requirements	Link
Technical Requirements	Link
Contracts	File: ./contracts/PaydeceEscrowV3.sol SHA3: 441c926486576e3adeb2f988ad5907c7898664b9b4c28812285b978e1c9624fa File: ./contracts/USDTToken.sol SHA3: acfa51095aa02b961837564b1bc4295015e2fd709747f3c6cb1f3f38014b9f2d

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation by external or internal actors.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation by external or internal actors.
Medium	Medium vulnerabilities are usually limited to state manipulations but cannot lead to asset loss. Major deviations from best practices are also in this category.
Low	Low vulnerabilities are related to outdated and unused code or minor Gas optimization. These issues won't have a significant impact on code execution but affect code quality

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

Code quality

The total Code Quality score is **10** out of **10**.

- The development environment is configured.
- The code is compliant with Solidity Code Style guide.

Test coverage

Code coverage of the project is **86.36%** (branch coverage).

- Code coverage is sufficient.

Security score

As a result of the audit, the code contains **no** issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.5**.

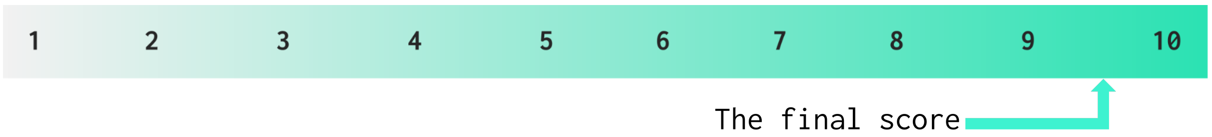

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
16 January 2023	2	1	2	0
3 February 2023	1	0	2	0
16 February 2023	0	0	0	0
1 March 2023	0	0	0	0

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Not Relevant
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed

Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Passed
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Not Relevant
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155 EIP-712	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Passed
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of Unused Variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP Standards Violation	EIP	EIP standards should not be violated.	Passed
Assets Integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed
User Balances Manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Passed

Token Supply Manipulation	Custom	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Not Relevant
Gas Limit and Loops	Custom	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Not Relevant
Style Guide Violation	Custom	Style guides and best practices should be followed.	Passed
Requirements Compliance	Custom	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	Custom	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	Custom	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
Tests Coverage	Custom	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed
Stable Imports	Custom	The code should not reference draft contracts, which may be changed in the future.	Passed

System Overview

Paydece is an escrow system with the following contracts:

- *PaydeceEscrowV3* – an escrow contract that allows creating new escrows of USDT tokens using other coins or native tokens.
- *USDTToken* – a USDT token for test purposes.

Privileged roles

- The owner of the *PaydeceEscrowV3* contract can set fees for buyers and sellers, release escrows, refund buyers, withdraw fees, add and delete addresses of tokens.

Risks

- The system can accept any ERC-20 token for escrow but is not designed to work with **fee-on-transfer tokens** and assume a value passed as a parameter as an actual value transferred. The project owners should ensure that such tokens are **never added to a whitelist**.

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

H01. Highly Permissive Role Access

The owner of the project can modify the project fee without any restrictions. Such permissions should be properly and in detail described in the documentation, so the users will be notified about such functionality.

Paths: ./contracts/PaydeceEscrowV3: setFeeSeller(); ./contracts/PaydeceEscrowV3: setFeeBuyer();

Recommendation: Add highly permissive functionality to the documentation.

Status: Fixed (The description was added to the documentation)

H02. Requirements Violation

Function *refundBuyerNativeCoin* is used to return buyer funds after a resolved dispute; however, it sends funds to a *_seller* instead of returning them to the *_buyer*.

Path: ./contracts/PaydeceEscrowV3: refundBuyerNativeCoin();

Recommendation: Refactor code to fit the requirements.

Status: Fixed (commit: fa22e1f)

H03. Unfinalized Code

The contract contains a lot of commented-out code and code that is used for debugging purposes like (*Log* event emits).

Due to emitting of redundant events, Gas consumption is increased.

Path: ./contracts/PaydeceEscrowV3

Recommendation: Clean up the code.

Status: Fixed (commit: 5127f70)

■■ Medium

M01. Usage of Built-In Transfer

The built-in transfer and send functions process a hard-coded amount of Gas. If the receiver is a contract with receive or fallback function, the transfer may fail due to the “out of Gas” exception.

Paths: ./contracts/PaydeceEscrowV3: _releaseEscrowNativeCoin(); ./contracts/PaydeceEscrowV3: refundBuyerNativeCoin(); ./contracts/PaydeceEscrowV3: withdrawFeesNativeCoin();

Recommendation: Replace transfer functions with call or provide special mechanism for interacting with a smart contract.

Status: Fixed (commit: fa22e1f)

■ Low

L01. Floating Pragma

Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Path: ./contracts/USDTToken.sol

Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

Status: Fixed (commit: 5127f70)

L02. Check-Effect-Interaction Pattern Violation

Contract violates the CEI pattern. In function *withdrawFees* some state variables are updated after the external calls.

Path: ./contracts/PaydeceEscrowV3: withdrawFees();

Recommendation: Refactor your code to fit CEI pattern.

Status: Fixed (commit: fa22e1f)

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

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 the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.