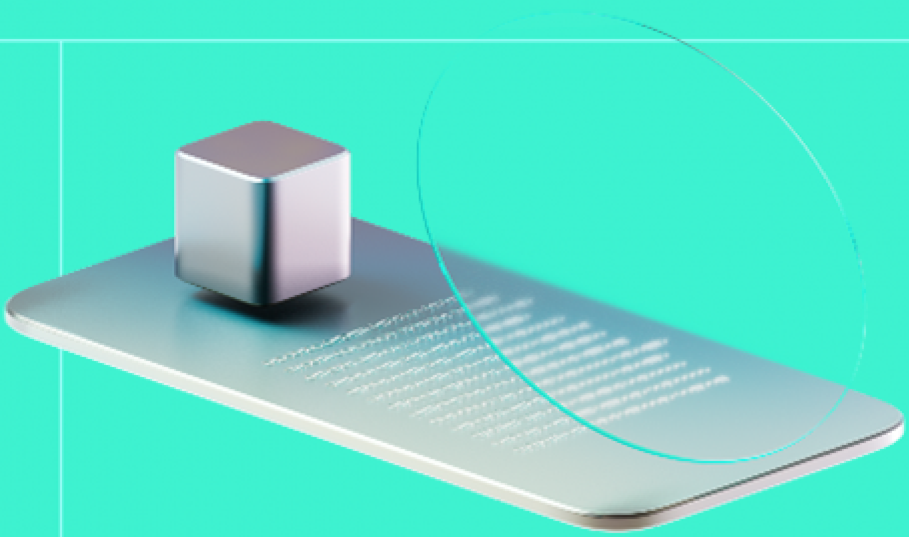
Smart Contract Code Review And Security Analysis Report



Customer: Upcade

Date: 24/01/2024

We express our gratitude to the Upcade team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

Upcade is a vault system integrating user deposits and withdrawals between blockchain and the Upcade gaming platform.

Platform: Polkadot

Language: Rust

Tags: Vault

Timeline: 17/01/2024 - 19/01/2024

Methodology: https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/GameDev-Tube/Upcade.Vault.A0
Commit	7f8c6e2

Audit Summary

10/10	8/10	100%	10/10
Security Score	Code quality score	Test coverage	Documentation quality score

Total 9.6/10

The system users should acknowledge all the risks summed up in the risks section of the report

1	0	1	0
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	0
High	0
Medium	0
Low	1

Vulnerability	Status
F-2024-0538 - Lack of On-Chain Record for Deposits and Centralized Reward Withdrawal Mechanism	Accepted

Document

Name	Smart Contract Code Review and Security Analysis Report for Upcade
Audited By	Eren Gonen
Approved By	Ataberk Yavuzer
Website	https://azero.upcade.xyz/
Changelog	22/01/2024 - Preliminary Report
	24/01/2024 - Final Report

Table of Contents

System Overview

Upcade is a vault where users can deposit tokens. The owners manage these tokens, rewards, and withdrawals through defined roles in the system, such as 'owner' and 'manager,' with the following contracts:

upcade_vault — The contract serves as a vault for user-deposited tokens, encompassing functions for withdrawals and rewards, which are controlled by specific roles within the system. Additionally, this contract features an access control mechanism to manage the addresses of owners and managers.

Privileged roles

- The owner of the contract can add, remove manager addresses.
- The owner has the capability to withdraw tokens to a specific address from the contract's balance.
- The owner has the capability to withdraw all tokens from the contract's balance.
- The owner and manager can distribute rewards from the contract's balance.

Executive Summary

This report presents an in-depth analysis and scoring of the customer's smart contract project. Detailed scoring criteria can be referenced in the scoring methodology.

Documentation quality

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

- Functional requirements are partially provided.
- Technical description is provided.

Code quality

The total Code Quality score is **8** out of **10**.

- The development environment is configured.
- The reward withdrawal mechanism is centralized.

Test coverage

Code coverage of the project is **100%**

- Deployment and user interactions are thoroughly tested, covering both negative and positive cases.

Security score

Upon auditing, the code was found to contain **0** critical, **0** high, **0** medium, and **1** low severity issues, leading to a security score of **10** out of **10**.

All identified issues are detailed in the “Findings” section of this report.

Summary

The comprehensive audit of the customer's smart contract yields an overall score of **9.6**. This score reflects the combined evaluation of documentation, code quality, test coverage, and security aspects of the project.

Risks

- The contract owner has the authority to withdraw all tokens deposited in the contract.
- The algorithm for the reward mechanism incorporates logic outside the contract - **which is out of the scope.**

Findings

Vulnerability Details

F-2024-0538 - Lack of On-Chain Record for Deposits and Centralized Reward Withdrawal Mechanism - Low

Description:

The contract does not maintain on-chain records of depositor addresses and their corresponding deposited amounts. Instead, it relies on off-chain storage for these significant pieces of information. This approach raises concerns regarding verifiability, and decentralization. In the absence of on-chain tracking, the contract lacks the capability to independently verify the correctness of reward addresses and amounts. The `deposit()` function, as shown, only emits events with deposit information rather than storing it on-chain.

```
pub fn deposit(&mut self, account_id: String, wallet_id: String) -> Result
<(), Error> {
// Get transferred native amount
let transferred_balance: u128 = self.env().transferred_value();

// Check if any tokens are transferred
if transferred_balance == 0 {
return Err(Error::NoFundsTransferred);
}

// Emit an event with the sender's address, transferred balance, and trans
action UUID
self.env().emit_event(Deposited {
from: self.env().caller(),
amount: transferred_balance,
account_id: account_id.clone(),
wallet_id: wallet_id.clone()
});

return Ok(());
}
```

Additionally, the reward withdrawal mechanism is controlled by the contract's owner and managers, as indicated in the `reward()` function.

```
#[ink(message)]
pub fn reward(
&mut self,
to: AccountId,
amount: Balance,
transaction_uuid: String,
) -> Result<(), Error> {
// Ensure that the caller is the owner or a manager
if !is_owner_or_manager(&self.access_control, self.env().caller()) {
return Err(Error::UnauthorizedCall);
}

// Ensure that the contract has enough balance to perform the reward
if self.env().balance() < amount {
return Err(Error::InsufficientBalance);
}

// Transfer the specified amount to the given address
match self.env().transfer(to, amount) {
Ok(()) => {
// Emit event
self.env().emit_event(Rewarded {
rewarded: to,
amount: amount,
uuid: transaction_uuid.clone(),
});
Ok(())
}
```

```
  _ => Err(Error::TransferFailed),
 }
 }
 }
```

Consequently, the contract lacks the capability to independently verify the correctness of deposited amounts and associated addresses. There is no function available for users to independently withdraw their deposited amounts or claim rewards. This design leads to a centralized mechanism for managing withdrawals and rewards, where such functions are controlled solely by the owner and managers, rather than being verifiable and enforceable by the contract's logic.

Impact:

- **User Trust:** Users may have less trust in a system where they cannot independently verify their balances on the blockchain.
- **Security and Integrity:** The integrity of the contract's financial operations could be compromised if off-chain data is inaccurate or manipulated.

Assets:

- lib.rs [https://github.com/GameDev-Tube/Upcade.Vault.A0]

Status: Accepted

Classification

Severity: Low

Impact: Likelihood [1-5]: 3

Impact [1-5]: 4

Exploitability [1,2]: 2

Complexity [0-2]: 0

Final Score: 2.3052181460292234 [Low]

Recommendations

Recommendation: Modify the contract to implement functionality for verifying the reward withdrawal address and amount. Additionally, enable users to withdraw their rewards from their deposited address.

Remediation: The Upcade team acknowledged this finding:

> When creating a game, our system locks the funds that players bet. If we allowed players to manage their rewards there could be fraud, e.g. someone lost but withdrew the funds of the whole bet to himself during the match, etc. In such a case, other players also lose their funds. This solution allows us to avoid this type of fraud and provides security for all sides of the game.

Observation Details

F-2024-0537 - Missing Events - Info

Description: Events for critical state changes should be emitted for tracking actions off-chain.

It was observed that events are missing events in the following functions:

```
add_manager()
remove_manager()
set_fee_recipient()
```

Events are crucial for tracking changes on the blockchain, especially for actions that alter significant contract states or permissions. The absence of events in these functions means that external entities, such as user interfaces or off-chain monitoring systems, cannot effectively track these important changes.

Assets:

- lib.rs [https://github.com/GameDev-Tube/Upcade.Vault.A0]

Status: Fixed

Recommendations

Recommendation: Consider implementing and emitting events for the necessary functions.

Remediation(7f8c6e2): The events were implemented and integrated into the necessary functions.

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.

Appendix 1. Severity Definitions

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
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.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details - initial

Repository	https://github.com/GameDev-Tube/Upcade.Vault.A0/
Commit	bbaf806336ecd5f374c06dcebf419a7ff2a0ce8e
Whitepaper	N/A
Requirements	README
Technical Requirements	README

Scope Details - second

Repository	https://github.com/GameDev-Tube/Upcade.Vault.A0/
Commit	7f8c6e2654ebf89b9cb21eab2831c46994deb2a2
Whitepaper	N/A
Requirements	README
Technical Requirements	README

Contracts in Scope

./contracts/lib.rs

./contracts/errors/upcade_errors.rs

./contracts/errors/mod.rs

./contracts/extensions/access_control.rs

./contracts/extensions/mod.rs