SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Enjinstarter
Date: September 9th, 2022

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for Enjinstarter
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	Staking
Platform	EVM
Network	Ethereum
Language	Solidity
Methods	Manual Review, Automated Review, Architecture Review
Website	https://enjinstarter.com/
Timeline	10.08.2022 - 09.09.2022
Changelog	17.08.2022 - Initial Review 01.09.2022 - Second Review 09.09.2022 - Third Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Enjinstarter (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/enjinstarter/ejs-staking-contracts
Commit:
d851e0e9c34540838a9bfd1bc1149aa33d84a239
Documentation:

Technical description

Integration and Unit Tests: Yes
Contracts:

File: ./contracts/AdminPrivileges.sol
SHA3: d916422c04a2d2968e365cf6f17a25d8b2c8add07e387c2a0f4fa5d39a8d9f49

File: ./contracts/AdminWallet.sol
SHA3: b33fb08ef971b80b4dfed870f1d3d95ac6db416f7a01e30e40390c5bdb18d968

File: ./contracts/interfaces/IAdminWallet.sol
SHA3: 34d03bceb310f1d8bbc695da5149e1fb1be7202016a96a2c062a47bd2a76d9b1

File: ./contracts/interfaces/IStakingPool.sol
SHA3: 5413c4c8f95d4f1876b89e935655fea77ccfcaa2b4d022e581fdaefe691e1af0

File: ./contracts/interfaces/IStakingService.sol
SHA3: c62b37daa6c46d907c9d11046060b33d790828bb79cd9309a6df237aba8304fc

File: ./contracts/libraries/UnitConverter.sol
SHA3: f7b33b391016f11e2617c0104c57a31eeb9a7451a231b8b714edc15054bf18dc

File: ./contracts/StakingPool.sol
SHA3: 51002f7638859a535b21a164e1da9507c7e9d406ea8ea79fdf987b89f1f3c887

File: ./contracts/StakingService.sol
SHA3: a0a77cb0c507b50610e7662d3a844e82ef7d22406874b74164a9efa414a3c217

Second review scope

Repository:
https://github.com/enjinstarter/ejs-staking-contracts
Commit:
53de193ee5483c0d5a1a35d12104478494cdce69
Documentation:

Technical description

Integration and Unit Tests: Yes
Contracts:

File: ./contracts/AdminPrivileges.sol
SHA3: d916422c04a2d2968e365cf6f17a25d8b2c8add07e387c2a0f4fa5d39a8d9f49

File: ./contracts/AdminWallet.sol
SHA3: b33fb08ef971b80b4dfed870f1d3d95ac6db416f7a01e30e40390c5bdb18d968

File: ./contracts/interfaces/IAdminWallet.sol
SHA3: 34d03bceb310f1d8bbc695da5149e1fb1be7202016a96a2c062a47bd2a76d9b1

File: ./contracts/interfaces/IStakingPool.sol
SHA3: 5413c4c8f95d4f1876b89e935655fea77ccfcaa2b4d022e581fdaefe691e1af0

File: ./contracts/interfaces/IStakingService.sol
SHA3: c62b37daa6c46d907c9d11046060b33d790828bb79cd9309a6df237aba8304fc

File: ./contracts/libraries/UnitConverter.sol
SHA3: f7b33b391016f11e2617c0104c57a31eeb9a7451a231b8b714edc15054bf18dc

File: ./contracts/StakingPool.sol
SHA3: 51002f7638859a535b21a164e1da9507c7e9d406ea8ea79fdf987b89f1f3c887

File: ./contracts/StakingService.sol
SHA3: a0a77cb0c507b50610e7662d3a844e82ef7d22406874b74164a9efa414a3c217

Third review scope

Repository:

https://github.com/enjinstarter/ejs-staking-contracts

Commit:

b3732b66037dcbe43f82e79678b43698328210d4

Documentation:

Technical description

Integration and Unit Tests: Yes

Contracts:

File: ./contracts/AdminPrivileges.sol
SHA3: f06109c968d9b00b84f77d3b715d04a92d58cdbab19b53f7b5bff45f7aa91a86

File: ./contracts/AdminWallet.sol
SHA3: 867f4b3f94f1f849145cea04ae4835cbd790eb94f0f5e284f72ae94207ee1f3e

File: ./contracts/interfaces/IAdminWallet.sol
SHA3: 26337a549da4a9eed9f153d8586a0ea305ac80d1931296d5dae54be03751bb44

File: ./contracts/interfaces/IStakingPool.sol
SHA3: 2aa3aa061112bf69f15da5128176ca2173af4e878756fea0d3f7af5287dddb56

File: ./contracts/interfaces/IStakingService.sol
SHA3: 2c8bc8dd5fd3ceee4d91490dd551e9021636e4593b19689c4ce5b5d710a8ae6a

File: ./contracts/libraries/UnitConverter.sol
SHA3: 5cc18c5d5d8699d3dbdd4605e97e48cbffb24e59d4bc88e18ceaabb94b240213

File: ./contracts/StakingPool.sol
SHA3: 15d0249f6e9348b3a3e6cce12a79fb3bb9578ec0e3334e26a1d7a9bd00617064

File: ./contracts/StakingService.sol
SHA3: eb0525ecf715caa7d364b871b53767c4028bdbf9518f0ff5d54b4d3d0941c964

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions.
Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution.

Executive Summary

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

Documentation quality

The total Documentation Quality score is **10** out of **10**. Functional and technical requirements were provided. Code is followed by NatSpec comments.

Code quality

The total CodeQuality score is **10** out of **10**. Code follows the Style guide. Deployment and basic user interactions are covered with tests. **Test coverage is 100%.**

Architecture quality

The architecture quality score is **10** out of **10**. Contracts, in general, use best practices.

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: **10.0**.

1
2
3
4
5
6
7
8
9
10
The final score

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
15 August 2022	2	0	1	0
1 September 2022	0	0	1	0
9 September 2022	0	0	0	0

Checked Items

We have audited provided smart contracts for commonly known and more specific vulnerabilities. Here are some of the items that are 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.	Not Relevant
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.	Failed
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 it is required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Authorization	SWC-115	tx.origin should not be used for	Passed

through tx.origin		authorization.	
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	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifier should always be used. All parameters from the signature should be used in signer recovery	Not Relevant
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.	Not Relevant
Assets integrity	Custom	Funds are protected and cannot be withdrawn without proper permissions.	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.	Not Relevant
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	Passed

		data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	
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 100%, 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, that may be changed in the future.	Passed

System Overview

Enjinstarter creates multiple staking pools using smart contracts for our users to earn rewards by locking up specific ERC20 tokens for a specified duration.

- StakingService.sol - stores the stake data and runtime staking pool data
- StakingPool.sol - stores the staking pool configuration information that is used by the staking service contract
- IStakingService.sol - interface for StakingService
- IStakingPool.sol - interface for StakingPool
- UnitConverter.sol - provides utility functions to convert between amounts specified in Wei and decimals
- AdminWallet.sol - provides an implementation of the admin wallet interface that is inherited by other contracts
- AdminPrivileges.sol - provides the role definitions that are inherited by other contracts
- IAdminWallet.sol - interface for AdminWallet

Privileged roles

- Default Admin: Admin role that controls the granting of roles to and revoking roles from accounts
- Governance: Controls access to functions that may result in loss of funds, such as the change of staking pool contract address, the change of admin wallet to receive revoked stakes and unused rewards and pause/unpause contract
- Contract Admin: Controls access to functions for day-to-day operations, such as creating staking pool, adding staking pool reward, opening/closing staking pool, suspending/resuming staking pool, suspending/resuming stake, and revoking stake
- User: Can stake funds, claim rewards, and unstake funds in available staking pools

Risks

- Contract admins can change staking pool contract address, change admin wallet, pause and unpause contract, close staking pools, revoke stakes, and withdraw rewards. In case of a governance fund keys leak, an attacker can get access to funds that belong to users.

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

1. Highly permissive owner access

The owner can change the staking pool contract address, change the admin wallet, pause and unpause the contract, close staking pools, suspend and resume staking pools and stakes. Such functionality is critical and should be *publicly* described, so the users will be notified about it.

Path: ./contracts/StakingService.sol, ./contracts/StakingPool.sol

Recommendation: Add highly permissive functionality to the *public* documentation.

Documentation: Link

Status: Mitigated (Documentation link provided)

■■ Medium

No medium severity issues were found.

■ Low

2. Possible code simplification

Some *require* statements in code are repeated multiple times in different functions (staking pool info check). This leaves code duplications and makes code harder to read and refactor.

Path: ./contracts/StakingService.sol

Recommendation: Separate code repetition in the separated function.

Status: Fixed (Revised commit: 53de193)

3. 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.

Paths: ./contracts/StakingService.sol, ./contracts/StakingPool.sol, ./contracts/AdminWallet.sol, ./contracts/AdminPrivileges.sol, ./contracts/libraries/StakingPool.sol, ./contracts/interfaces/IStakingPool.sol, ./contracts/interfaces/IAdminWallet.sol, ./contracts/interfaces/IStakingService.sol

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

Customer’s notice: Due to business requirement to deploy smart contracts to multiple different EVM blockchains, we are unable to lock the compiler version as we may need to use different compiler version.

Status: Mitigated (with Customer notice)

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date 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 to 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, Consultant cannot guarantee the explicit security of the audited smart contracts.