HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: TheNextWar
Date: May 25th, 2022

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for TheNextWar
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type	ERC20 token; Staking
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	https://www.thenextwar.io
Timeline	16.05.2022 - 25.05.2022
Changelog	16.05.2022 - Initial Review 25.05.2022 - Second Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Introduction 4

Scope 4

Severity Definitions 6

Executive Summary 7

Checked Items 8

System Overview 11

Findings 12

Disclaimers 15

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by TheNextWar (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/TheNextWar/Contracts
Commit:
334fe81ea7282ac4f768de077c37a7932c0302de
Technical Documentation:
Type: Whitepaper
Link: https://thenextwar.gitbook.io/the-next-war/the-next-war-platform/introduction

JS tests: Yes
Contracts:
File: ./contracts/Distribution.sol
SHA3: 856f885e0adbf791533d85594e7b48701c2cfa0be997875525475c0d165954e6

File: ./contracts/TngToken.sol
SHA3: 4f74a1b8f68b7806a71b57ae72a35bd2d32dda4aa03ab8342d8a009718625cbd

File: ./contracts/TncToken.sol
SHA3: c6fea0b7ff8c9cff1ef9c0bb892de123bd1043b8d9c39761c7fcf53781b0a1e1

File: ./contracts/Staking.sol
SHA3: 8612e9bbbf9f17a3e5349cb1ab6f15d4a469bc6480fc68de51252fc2e0dba9d6

Second review scope

Repository:
https://github.com/TheNextWar/Contracts
Commit:
02e100d34073a279c76c59ff00aa650e3936bf83

Technical Documentation:
Type: Whitepaper
Link: https://thenextwar.gitbook.io/the-next-war/the-next-war-platform/introduction

JS tests: Yes
Contracts:
File: ./contracts/Distribution.sol
SHA3: d977fcac01ecfd08d1109ed5dec0ab106b6e747c56ce066003d6d115e7f2f685

File: ./contracts/TngToken.sol
SHA3: d900bc895da2a00c912cee8495b213a7a6dc02066be712cfc64473b96592f1a6

File: ./contracts/TncToken.sol
SHA3: 3a2cf8b7201a248312585afba67c0b41949e10d3b914965fb5869fb0d5a61323

File: ./contracts/Staking.sol

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

SHA3: 8430d9f42a07edcb1a581b2490bac70bdbdd96ef928a4d067a773bd1ca137d7a

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Executive Summary

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

Documentation quality

The Customer provided a whitepaper with technical documentation that includes functional requirements. The total Documentation Quality score is **10** out of **10**.

Code quality

The total CodeQuality score is **10** out of **10**. Code is well structured and has a good unit tests coverage and many comments.

Architecture quality

The architecture quality score is **10** out of **10**. The logic is separated for different files, following a single responsibility principle.

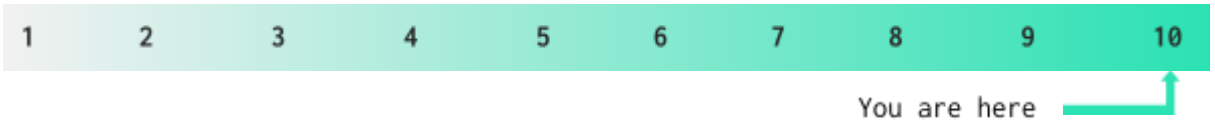
Security score

As a result of the audit, security engineers found 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**.



Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Not Relevant
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 destroyed until it has funds belonging to users.	Passed
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Uninitialized Storage Pointer	SWC-109	Storage type should be set explicitly if the compiler version is < 0.5.0.	Not Relevant
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Not Relevant
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.	Passed
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 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.	Passed
Signature Unique Id	SWC-117 SWC-121 SWC-122	Signed messages should always have a unique id. A transaction hash should not be used as a unique id.	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.	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.	Passed
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.	Passed

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
Repository Consistency	**Custom**	The repository should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
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

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

System Overview

THE NEXT WAR is a blockchain-based Battle Royale inspired by Call of Duty. A shooter game featuring world-class tournaments bringing shooter enthusiasts to a whole new level of extreme gaming. The system has the following contracts:

- TngToken — a simple ERC-20 token that mints all initial supply to a specified during deploy address. Additional minting is not allowed.
  It has the following attributes:
  - Name: THE NEXT WAR GEM
  - Symbol: TNG
  - Decimals: 18
  - Total supply: 5b (token supply could not be verified before the contract deploy).
- TncToken — simple ERC-20 token with not limited minting.
  It has the following attributes:
  - Name: THE NEXT WAR COIN
  - Symbol: TNC
  - Decimals: 18
  - Total supply: Unlimited.
- Staking — a contract that rewards users for staking their tokens.
- Distribution - a contract that is responsible for token allocation distribution to users. Users can claim allocated tokens according to the vesting rules.

Privileged roles

- The owner of the *Distribution* contract can allocate tokens for a specific address
- The owner of the *Distribution* contract can reset token allocation for a specific address
- The owner of the *Distribution* contract can change the vesting period and percentage per vesting round
- The owner of the *Distribution* contract can transfer any token sent to the contract
- The owner of the *Staking* contract can change emergency withdrawal fee. The max fee is limited to 20%
- The owner of the *Staking* contract can change the staking lock period
- The owner of the *Staking* contract can change the staking reward amount

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Findings

Critical

1. **Inconsistent data.**

   The `deposit` function checks the balance of TNG tokens, but the user deposits LP tokens instead. As a result - TNG tokens are not used during the deposit but are required to be on the user account for deposit operation. The same is applicable for the `harvest` function as well.

   Contracts: Staking.sol

   Function: deposit, harvest

   Recommendation: Check the logic of the deposit function. If it is correct - it is recommended to highlight in documentation such behavior.

   Status: Fixed (02e100d34073a279c76c59ff00aa650e3936bf83)

High

1. **Missing validation.**

   The function allows to withdraw any tokens from the contracts.

   Funds that belong to users can be affected.

   Contract: Staking.sol

   Function: rescueToken

   Recommendation: Ensure that user's funds will not be withdrawn using this method. If `_token` is equal to the staking token, the contract should always have a balance equal to `lpTokenDeposited` + `pendingTngRewards`.

   Status: Fixed (02e100d34073a279c76c59ff00aa650e3936bf83)

2. **Insufficient rewards balance.**

   Withdraw function returns to the user originally staked funds and pays TNG reward. In case when the contract does not have a TNG token - withdraw function would be blocked, and the user would not be able to return originally staked funds.

   Contracts: Staking.sol

   Function: withdraw

   Recommendation: Implement the `emergencyWithdraw` function, which returns staked funds without rewards.

   Status: Fixed (02e100d34073a279c76c59ff00aa650e3936bf83)

3. **Stable imports.**

The code should not reference draft contracts that may be changed in the future.

The current implementation of Staking and Distribution contracts allows updating TNG token address after contract deployment.

Contracts: Staking.sol, Distribution.sol

Function: setTngToken

Recommendation: Remove the ability to update the TNG token address after the deployment.

Status: Fixed (02e100d34073a279c76c59ff00aa650e3936bf83)

Medium

1. **Unnecessary SafeMath usage.**

   Solitidy >= 0.8.0 provides errors for buffer overflow and underflow. No need to use SafeMath anymore.

   Recommendation: Do not use SafeMath.

   Status: Fixed (02e100d34073a279c76c59ff00aa650e3936bf83)

2. **TncToken access loss is possible.**

   `authorize` method of TNC token is available only to the owner and authorized user, which by default is true. The owner is able to reset his authorization, and in this case, mint access would be lost.

   Contracts: TncToken.sol

   Functions: authorize

   Recommendation: Check if the logic of how `authorize` is implemented and `require(isAuthorized)` is not a mistake.

   Status: Fixed (02e100d34073a279c76c59ff00aa650e3936bf83)

3. **Unchecked transfer.**

   The return value of an external transfer/transferFrom call is not checked. Several tokens do not revert in case of failure and return false. If one of these tokens is used in Staking, the deposit will not revert if the transfer fails, and an attacker can call the deposit for free.

   Contracts: Staking.sol, Distribution.sol

   Functions: deposit, withdraw, payTngReward, claim, rescueToken

   Recommendation: Check the result of the transfer if it is true.

   Status: Fixed (02e100d34073a279c76c59ff00aa650e3936bf83)

Low

1. **Variable Shadowing.**

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Solidity allows for ambiguous naming of state variables when inheritance is used. Contract A with a variable x could inherit contract B, which has a state variable x defined. This would result in two separate versions of x, accessed from contract A and the other from contract B. In more complex contract systems, this condition could go unnoticed and subsequently lead to security issues.

Contracts: TngToken.sol, TncToken.sol,

Functions: TncToken.constructor(string **name**) -> ERC20.**name**(),
TncToken.constructor(string **symbol**) -> ERC20.**symbol**(),
TncToken.constructor(uint256 **totalSupply**) -> ERC20.**totalSupply**(),
TngToken.constructor(string **name**) -> ERC20.**name**(),
TngToken.constructor(string **symbol**) -> ERC20.**symbol**(),
TngToken.constructor(uint256 **totalSupply**) -> ERC20.**totalSupply**()

Recommendation: Consider renaming the function argument.

Status: Fixed (02e100d34073a279c76c59ff00aa650e3936bf83)

2. **The public function could be declared external.**

Public functions that are never called by the contract should be declared external to save Gas.

Contracts: Distribution.sol

Functions: getClaimableAmount

Recommendation: Use the external attribute for functions never called from the contract.

Status: Fixed (02e100d34073a279c76c59ff00aa650e3936bf83)

3. **Missing events arithmetic**

To simplify off-chain changes tracking, it is recommended to emit events when a crucial part of the contract changes.

Contracts: Distribution.sol, Staking.sol

Functions: setLockTime, setClaimable, setTngPerSecond

Recommendation: Emit an event for critical parameter changes.

Status: Fixed (02e100d34073a279c76c59ff00aa650e3936bf83)

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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 audit makes no statements or warranties on the security of the code. It also cannot be considered a 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.

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 audit cannot guarantee the explicit security of the audited smart contracts.