HACKEN

# SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

**Customer**: Gunzilla
**Date**: May 5, 2023

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 Gunzilla |
| **Approved By** | Paul Fomichov \| Lead Solidity SC Auditor at Hacken OU |
| **Type** | ERC20 token; Vesting; |
| **Platform** | EVM |
| **Language** | Solidity |
| **Methodology** | Link |
| **Website** | https://gunzillagames.com/en/ |
| **Changelog** | 11.04.2023 - Initial Review<br>05.05.2023 - Second Review |

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

# Table of contents

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

## Introduction

Hacken OÜ (Consultant) was contracted by Gunzilla (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 includes the following smart contracts from the provided repository:

### Initial review scope

| | |
|---|---|
| **Repository** | https://github.com/Gunzilla-Games/tokenomics-smart-contract |
| **Commit** | afe95e57 |
| **Whitepaper** | - |
| **Functional Requirements** | https://github.com/Gunzilla-Games/tokenomics-smart-contract/blob/main/README.md |
| **Technical Requirements** | - |
| **Contracts** | File: ./contracts/GunzillaFixedDatePaymentPlan.sol<br>SHA3: 5d121cc49307e9acbb8ec401f879ef735900b93655294cf05a0b306503ec5290<br><br>File: ./contracts/GunzillaLinearPaymentPlan.sol<br>SHA3: 1cc94e23504f7b01ec1ab167b2a140e77ef6ae40e87f9d77673aa0b92ecf11bc<br><br>File: ./contracts/GunzillaPaymentPlan.sol<br>SHA3: dbd394abc9648ec39daeb057117665a333872c2a2dae4e954c0f8d1f33184a3c<br><br>File: ./contracts/GunzillaToken.sol<br>SHA3: d0e6584039053663733da12cb509095595823e67e8da2d09eed7a8fed3b2e474<br><br>File: ./contracts/WGUN.sol<br>SHA3: 247a3f92aa19a07bc79fedfaaf9e9fd4e9abfe0156cfa286c5befade7a28d851<br><br>File: ./contracts/interfaces/IBotProtector.sol<br>SHA3: 422682ad4a026a887b236146f7606e4c983db60b8bc6b44adf00e9667966182e |

### Second review scope

| | |
|---|---|
| **Repository** | https://github.com/Gunzilla-Games/tokenomics-smart-contract |
| **Commit** | 70668c06a |
| **Whitepaper** | - |
| **Functional Requirements** | https://github.com/Gunzilla-Games/tokenomics-smart-contract/blob/main/README.md |

| | |
|---|---|
| **Technical Requirements** | https://vakhtanhs-organization.gitbook.io/guzilla-gun-tokenomics/ |
| **Contracts** | File: ./contracts/GunzillaFixedDatePaymentPlan.sol<br>SHA3: bd0683caef890e159a8802e07791e5fcd3de26983b1d6e6ec984d84c58342376<br><br>File: ./contracts/GunzillaLinearPaymentPlan.sol<br>SHA3: e2b050e15ce85c92011d92bebcc23e854bfb5754917e597c4f655bb08fc15061<br><br>File: ./contracts/GunzillaPaymentPlan.sol<br>SHA3: 425c6f8c3695802760c7c4d0c3812fea3f3074d407e2d4f5ec8328acc78fa54b<br><br>File: ./contracts/GunzillaToken.sol<br>SHA3: 86d3adf234a88c994f2a2128cf4950c7121767f2079509b2fc93d96d5c2b3e12<br><br>File: ./contracts/WGUN.sol<br>SHA3: 91af46fa564d0b511c5a0149a16f165f4e442e5c76d6b99a533ec09e01421e8d<br><br>File: ./contracts/interfaces/IBotProtector.sol<br>SHA3: 422682ad4a026a887b236146f7606e4c983db60b8bc6b44adf00e9667966182e |

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 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 |

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 scoring methodology.

## Documentation quality

The total Documentation Quality score is **9** out of **10**.

- NatSpec is provided with a full repository description.
- No whitepaper.
- Run instructions are provided.
- Technical specification is provided.
- Functional requirements are provided.

## Code quality

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

- Solidity Style Guide violations.
- Best practice violations.

## Test coverage

Code coverage of the project is **100.00%** (branch coverage).

- Deployment and basic user interactions are covered with tests.

## Security score

As a result of the audit, the code contains **4** low severity 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**.

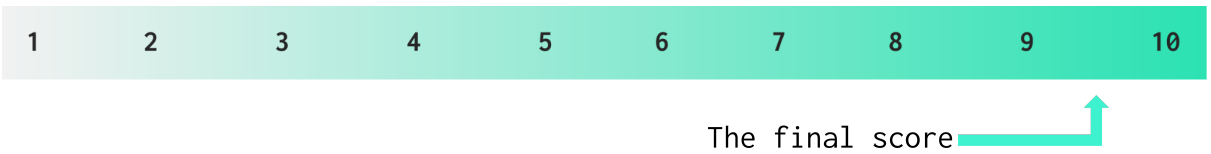


***Table. The distribution of issues during the audit***

| Review date | Low | Medium | High | Critical |
|---|---|---|---|---|
| 11 April 2023 | 6 | 1 | 2 | 0 |
| 27 April 2023 | 4 | 0 | 0 | 0 |

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

## Risks

- The smart contracts *WGUN, GunzillaToken* call functions from the external *BotProtector* smart contract, which is out of the audit scope. Its functionality cannot be verified.
- The logic of the `vestTokens()` function allows to provide tokens instantly if the owners of the smart contract set the appropriate vesting settings.

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

## System Overview

*Gunzilla* is a mixed-purpose system with the following contracts:

- *GunzillaToken* – ERC-20 token that mints all initial supply to an address passed during initialization. It is an upgradable, pausable smart contract.
  It has the following attributes:
  - Name: Gunzilla Token
  - Symbol: GUN
  - Decimals: 18
  - Total supply: 10b tokens.

- *WGUN* – ERC-20 token, which is wrap for the *GunzillaToken* It is an upgradable, pausable smart contract. Tokens can be minted in an amount equal to the amount Ether sent to the smart contract. Tokens can be burned by any holder, amount of Ether equal to burned amount would be transferred to the user.
  It has the following attributes:
  - Name: Wrapped Gunzilla Token
  - Symbol: WGUN
  - Decimals: 18
- *GunzillaPaymentPlan* - an abstract upgradable smart contract that is used for creating vesting for system users.
- *GunzillaLinearPaymentPlan* - a smart contract inherited from the *GunzillaPaymentPlan.* This smart contract uses linear token vesting. The cliff and periods duration and amount can be set by the owner.
- *GunzillaFixedDatePaymentPlan* - a smart contract inherited from the *GunzillaPaymentPlan.* This smart contract uses fixed dates to unlock tokens for certain users. The unlock schedule can be set by the owner.
- *IBotProtector* - an interface for using BotProtector smart contract functionality.

### Privileged roles

- The admin of the *GunzillaToken* contract can set the *BotProtector* variable.
- The PAUSE_ROLE of the *GunzillaToken* contract can pause/unpause smart contracts.
- The admin of the *WGUN* contract can set the *BotProtector* variable.
- The PAUSE_ROLE of the *WGUN* contract can pause/unpause smart contracts.
- The owner of the *GunzillaPaymentPlan* can pause/unpause smart contracts. The owner can set vesting settings for the user.

- The owner of the *GunzillaLinearPaymentPlan* is available to set payment plans for the vesting.
- The owner of the *GunzillaFixedDatePaymentPlan* is available to set payment plans for the vesting.

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

## 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. | Failed |
| **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 |

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

| | | | |
|---|---|---|---|
| **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. | Not Relevant |
| **Block values as a proxy for time** | SWC-116 | Block numbers should not be used for time calculations. | Passed |
| **Signature Unique Id** | SWC-117<br>SWC-121<br>SWC-122<br>EIP-155<br>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. | 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<br>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 |

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

| | | | |
|---|---|---|---|
| **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. | Failed |
| **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 |

# Findings

## ■■■■ Critical

No critical severity issues were found.

## ■■■ High

### H01. Unverifiable Logic

Some contracts in the scope interact with external smart contract *BotProtector* which is out of audit scope and can not be verified.

This leads to unexpected code behavior.

**Path:**
./contracts/GunzillaToken.sol : _transfer(),
./contracts/WGUN.sol : _transfer(),

**Recommendation**: Add the described functionality to the code or describe all used third-party smart contracts in the documentation with the appropriate links.

**Found in:** afe95e57

**Status**: Mitigated (Revised commit: 70668c06a)

### H02. Data Consistency

The owner passed as an argument in the *vestTokens()* function *_startDate* parameter, which will be used later for reward calculation. According to the logic of smart contract execution, this date can be set to any date (for example, 10 years ago).

This leads to the fact that all tokens can be withdrawn immediately after the *vestTokens()* function calls, which contradicts the smart contract logic.

**Path:**
./contracts/GunzillaPaymentPlan.sol : vestTokens()

**Recommendation**: Check the time frame inside the *vestTokens()* function.

**Found in:** afe95e57

**Status**: Mitigated (with Customer notice: The client stated they need this functionality to provide their services.)

## ■■ Medium

### M01. Missing Event for Critical Value Update

Critical state changes should emit events for tracking things off-chain.

The functions do not emit an event on change of important *botProtector* or *farmingContract*.

**Paths:**

./contracts/GunzillaPaymentPlan.sol : setFarmingContract(),

./contracts/WGUN.sol : setBotProtector(),

./contracts/GunzillaToken.sol : setBotProtector(),

**Recommendation**: Emit events on critical state changes mentioned above.

**Found in:** afe95e57

**Status**: Fixed (Revised commit: 70668c06a)

## Low

### L01. Missing Zero Address Validation

Address parameters are used without checking against the possibility of 0x0.

**Paths:**
./contracts/GunzillaToken.sol : initialize(), setBotProtector(),
./contracts/WGUN.sol : initialize(), setBotProtector(),
./contracts/GunzillaPaymentPlan.sol : setFarmingContract(), vestTokens(),
./contracts/GunzillaLinearPaymentPlan.sol : withdrawableAmount(),
./contracts/GunzillaFixedDatePaymentPlan.sol : withdrawableAmount()

**Recommendation**: Implement zero address checks.

**Found in:** afe95e57

**Status**: Reported (Revised commit: 70668c06a. Issue left in GunzillaPaymentPlan.sol: vestTokens())

### L02. Floating Pragma

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to 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/*

**Recommendation**: Pock the Solidity pragma version. Find more: SWC-103.

**Found in:** afe95e57

**Status**: Fixed (Revised commit: 70668c06a)

### L03. Style Guide: Maximum Line Length

The provided projects should follow the official guidelines. Maximum suggested line length 120 is not followed.

**Paths:**

./contracts/GunzillaToken.sol
./contracts/WGUN.sol
./contracts/GunzillaPaymentPlan.sol
./contracts/GunzillaLinearPaymentPlan.sol
./contracts/GunzillaFixedDatePaymentPlan.sol

**Recommendation**: Follow the official Solidity guidelines.

**Found in:** afe95e57

**Status**: Reported (Revised commit: 70668c06a. Issues left in:
./contracts/GunzillaLinearPaymentPlan.sol : 65, 71;
./contracts/GunzillaFixedDatePaymentPlan.sol : 50, 52;)

### L04. Use of Hard-Coded Values

Hard-coded values are used in computations.

**Path:**

./contracts/GunzillaToken.sol : initialize()

**Recommendation**: Convert these variables into constants.

**Found in:** afe95e57

**Status**: Reported (Revised commit: 70668c06a)

### L05. Unnecessary State Variable

All variables that are used in the code must have a rationale and must be used in the code for certain interactions. farmingContract, farmingContractPool state variable never used in code for any calculations or business logic operations.

This leads to inefficient use of Gas.

**Path:**
./contracts/GunzillaPaymentPlan.sol : farmingContract, farmingContractPool

**Recommendation**: Remove unnecessary variables.

**Found in:** afe95e57

**Status**: Fixed (Revised commit: 70668c06a)

### L06. Functions that Can Be Declared External

In order to save Gas, public functions that are never called in the contract should be declared as external.

**Paths:**

./contracts/GunzillaToken.sol : initialize(), pause(), unpause(),

./contracts/GunzillaPaymentPlan.sol : initialize(), pause(), unpause(), setFarmingContract(), vestTokens(), withdrawAvailableTokens(), withdraw(), vestingData(),

./contracts/WGUN.sol : vestingData(), withdraw(), pause(), unpause(), setBotProtector(),

./contracts/GunzillaFixedDatePaymentPlan.sol : paymentPlanData(), paymentPlanLength(),

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

**Found in:** afe95e57

**Status**: Fixed (Revised commit: 70668c06a)

### L07. Functions that Can Be Declared External

In order to save Gas, public functions that are never called in the contract should be declared as external.

**Paths:**

./contracts/GunzillaToken.sol : setBotProtector(),

./contracts/GunzillaLinearPaymentPlan.sol : addPaymentPlan(), paymentPlanData()

./contracts/GunzillaFixedDatePaymentPlan.sol : addPaymentPlan(),

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

**Found in:** 70668c06a

**Status**: New

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 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.