



**HACKEN**

# SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

**Customer:** G4AL

**Date:** March 15, 2023

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

<b>Name</b>	Smart Contract Code Review and Security Analysis Report for G4AL
<b>Approved By</b>	Yevheniy Bezuhlyi   SC Audits Head at Hacken OU
<b>Type</b>	Vesting
<b>Platform</b>	EVM
<b>Language</b>	Solidity
<b>Methodology</b>	<a href="#">Link</a>
<b>Website</b>	<a href="https://gamesforaliving.com/">https://gamesforaliving.com/</a>
<b>Changelog</b>	08.03.2023 - Initial Review 15.03.2023 - Second Review

## Table of contents

<b>Introduction</b>	<b>4</b>
<b>Scope</b>	<b>4</b>
<b>Severity Definitions</b>	<b>5</b>
<b>Executive Summary</b>	<b>6</b>
<b>Risks</b>	<b>7</b>
<b>System Overview</b>	<b>8</b>
<b>Checked Items</b>	<b>9</b>
<b>Findings</b>	<b>12</b>
Critical	12
C01. Denial of Service / Funds Lock	12
High	12
H01. Requirements Violation	12
Medium	13
Low	13
L01. Floating Pragma	13
L02. Inefficient Gas Optimization	13
L03. Functions That Can Be Declared External	14
L04. State Variables Can Be Declared Immutable	14
<b>Disclaimers</b>	<b>15</b>

## Introduction

Hacken OÜ (Consultant) was contracted by G4AL(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

<b>Repository</b>	<a href="https://github.com/gamesforaliving/web3-contracts-bundle">https://github.com/gamesforaliving/web3-contracts-bundle</a>
<b>Commit</b>	c8865b9fa321af7c95ae2797c4be976fc198ba00
<b>Functional Requirements</b>	<a href="#">Link</a>
<b>Technical Requirements</b>	<a href="#">Link</a>
<b>Contracts</b>	File: ./contracts/vestings/VestingBasic.sol SHA3: d1f90494c40be5d2faf77ee97d815aeae3182ef268720fffe22b3f7a8ee28983

### Second review scope

<b>Repository</b>	<a href="https://github.com/gamesforaliving/web3-contracts-bundle">https://github.com/gamesforaliving/web3-contracts-bundle</a>
<b>Commit</b>	2813080762721cdb92e832b6a7df74974fe51fe3
<b>Functional Requirements</b>	<a href="#">Link</a>
<b>Technical Requirements</b>	<a href="#">Link</a>
<b>Contracts</b>	File: ./contracts/vestings/VestingBasic.sol SHA3: 4d602ae1b549b44da5b6a7c35d99d9dbe6996bf21eeddc841e31e30b8a51ec4e

## Severity Definitions

Risk Level	Description
<b>Critical</b>	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.
<b>High</b>	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.
<b>Medium</b>	Medium vulnerabilities are usually limited to state manipulations but cannot lead to asset loss. Major deviations from best practices are also in this category.
<b>Low</b>	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**.

- Functional requirements are present.
- Technical description is provided.
- NatSpec is present.

### Code quality

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

- Solidity Code Style guide is followed.
- Gas optimization can be better.

### Test coverage

Code coverage of the project is **9.38%** (branch coverage).

- Since the contract lines of code are less than 250, the code coverage will not affect the score.

### Security score

As a result of the audit, the code contains **3** 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: **9.8**.



*Table. The distribution of issues during the audit*

Review date	Low	Medium	High	Critical
8 March 2023	3	0	1	1
22 March 2023	3	0	0	0

## Risks

- The repository contains a code that is **out of the audit scope**. We may not guarantee the secureness of such contracts.
- The DEAULT\_ADMIN\_ROLE has the right to change the vesting collector address anytime they want.

## System Overview

The audited contract of GamesForaLiving is a basic vesting protocol that handles the vesting for a single vesting collector address with a specific vesting ERC20 token.

The vesting consists of specific, manually logged vesting periods and amounts, set by the deployer of the contract.

After the periods are set, “vesters” that have the vester role can initiate withdrawals of the vestings to the vesting collector address of the contract.

The files in the scope:

- **VestingBasic.sol:** The contract responsible for the vesting logic that also stores the vesting token, collector, unlock time for vesting, and vesting schedule.

## Privileged roles

Roles defined in the system:

- DEFAULT\_ADMIN\_ROLE: Sets the vesting token, collector, and unlock time on deployment. Sets the vesting schedule after deployment, only once. Can grant VESTER\_ROLE to addresses.
- VESTER\_ROLE: Can call the function to withdraw the vesting token according to the wheezing schedule, to the vesting collector address.

## Recommendations

- Declare functions that are not called within the contract as external.
- Implement a way to stop iterations when not needed.



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

<b>Race Conditions</b>	<a href="#">SWC-114</a>	Race Conditions and Transactions Order Dependency should not be possible.	Passed
<b>Authorization through tx.origin</b>	<a href="#">SWC-115</a>	tx.origin should not be used for authorization.	Not Relevant
<b>Block values as a proxy for time</b>	<a href="#">SWC-116</a>	Block numbers should not be used for time calculations.	Not Relevant
<b>Signature Unique Id</b>	<a href="#">SWC-117</a> <a href="#">SWC-121</a> <a href="#">SWC-122</a> <a href="#">EIP-155</a> <a href="#">EIP-712</a>	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
<b>Shadowing State Variable</b>	<a href="#">SWC-119</a>	State variables should not be shadowed.	Passed
<b>Weak Sources of Randomness</b>	<a href="#">SWC-120</a>	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
<b>Incorrect Inheritance Order</b>	<a href="#">SWC-125</a>	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Not Relevant
<b>Calls Only to Trusted Addresses</b>	<a href="#">EEA-Leve1-2</a> <a href="#">SWC-126</a>	All external calls should be performed only to trusted addresses.	Not Relevant
<b>Presence of Unused Variables</b>	<a href="#">SWC-131</a>	The code should not contain unused variables if this is not <a href="#">justified</a> by design.	Passed
<b>EIP Standards Violation</b>	<a href="#">EIP</a>	EIP standards should not be violated.	Not Relevant
<b>Assets Integrity</b>	Custom	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed
<b>User Balances Manipulation</b>	Custom	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
<b>Data Consistency</b>	Custom	Smart contract data should be consistent all over the data flow.	Passed

<b>Flashloan Attack</b>	<b>Custom</b>	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
<b>Token Supply Manipulation</b>	<b>Custom</b>	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Not Relevant
<b>Gas Limit and Loops</b>	<b>Custom</b>	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
<b>Style Guide Violation</b>	<b>Custom</b>	Style guides and best practices should be followed.	Passed
<b>Requirements Compliance</b>	<b>Custom</b>	The code should be compliant with the requirements provided by the Customer.	Passed
<b>Environment Consistency</b>	<b>Custom</b>	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
<b>Secure Oracles Usage</b>	<b>Custom</b>	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
<b>Tests Coverage</b>	<b>Custom</b>	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.	Failed
<b>Stable Imports</b>	<b>Custom</b>	The code should not reference draft contracts, which may be changed in the future.	Passed

## Findings

### Critical

#### C01. Denial of Service / Funds Lock

The `withdraw()` function uses a loop that depends on an uncapped array length.

If the “vester” waits too long before calling the `withdraw()` function and lets withdrawable vestings pile up, the transaction cost may get too large.

If the `vestingSchedule.length` is too large, this could also lead to the transaction cost being too large even if the “vester” does not pile vestings because the loop iterates through the whole array even if there are no vestings to withdraw yet.

The function transactions may fail due to inefficient Gas and cause denial of service, which would lead to funds being locked in the contract.

**Path:** `./contracts/vestings/VestingBasic.sol : withdraw()`

**Recommendation:** Divide the `withdraw()` function to multiple transactions by limiting the iteration number of the loop.

**Found in:** `c8865b9fa321af7c95ae2797c4be976fc198ba00`

**Status:** Fixed (Revised commit: 2813080)

### High

#### H01. Requirements Violation

It is stated in the documentation that addresses that have the “vester” role can call the `withdraw()` function to withdraw their vested tokens by the following quote;

*“Allows vesters to withdraw their vested tokens.”*

However, this is not implemented in the contract. The functionality of the contract only lets the “vesters” to call the `withdraw()` function to withdraw the vested ERC20 tokens to the `vestingCollector` address.

This can lead to unexpected behavior.

**Path:** `./contracts/vestings/VestingBasic.sol : withdraw()`

**Recommendation:** Either update the documentation for the vesting logic, or implement different vesting schedules for every individual “vester” address that can call the `withdraw()` function.

**Found in:** c8865b9fa321af7c95ae2797c4be976fc198ba00

**Status:** Fixed (Revised commit: 2813080)

## ■ ■ Medium

No medium severity issues were found.

## ■ Low

### L01. Floating Pragma

The project uses floating pragma `^0.8.17`.

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version which may include bugs that affect the system negatively.

**Path:** ./contracts/vestings/VestingBasic.sol

**Recommendation:** Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment. Consider known bugs (<https://github.com/ethereum/solidity/releases>) for the compiler version that is chosen.

**Found in:** c8865b9fa321af7c95ae2797c4be976fc198ba00

**Status:** Fixed (Revised commit: 2813080)

### L02. Inefficient Gas Optimization

In the loop used in the `withdraw()` function, the iteration continues even if there are no more vestings left to claim.

Implementing a way to break the loop when there are no more withdrawable vestings left would improve gas efficiency.

**Path:** ./contracts/vestings/VestingBasic.sol : `withdraw()`, `setVestingSchedule()`

**Recommendation:** When setting vesting schedules, consider checking the input array `when` so that it is in ascending order. This way, the loop in the `withdraw()` function can be stopped when the iteration `vestingSchedule[i].when` parameter is larger than the `block.timestamp`.

**Found in:** c8865b9fa321af7c95ae2797c4be976fc198ba00

**Status:** Reported (The for loop continues iteration until the end of `vestingSchedule` array length even if there are no more funds to claim.)

### L03. 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.

**Path:** `./contracts/vestings/VestingBasic.sol : setVestingSchedule(), withdraw()`

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

**Found in:** `c8865b9fa321af7c95ae2797c4be976fc198ba00`

**Status:** **Reported** (Public functions are used even though they are not being called within the contract.)

### L04. State Variables Can Be Declared Immutable

Variables' `vestingScheduleMaxLength`, `unlockTime`, and `vestingToken` values are set in the constructor. These variables can be declared immutable.

This will lower the Gas cost.

**Path:** `./contracts/vestings/VestingBasic.sol : constructor()`

**Recommendation:** Declare mentioned variables as immutable.

**Found in:** `2813080762721cdb92e832b6a7df74974fe51fe3`

**Status:** **New**

## 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.