HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: G4AL
Date: March 15, 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 G4AL
Approved By	Yevheniy Bezuhlyi | SC Audits Head at Hacken OU
Type	Vesting
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://gamesforaliving.com/
Changelog	08.03.2023 - Initial Review 15.03.2023 - 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 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

Repository	https://github.com/gamesforaliving/web3-contracts-bundle
Commit	c8865b9fa321af7c95ae2797c4be976fc198ba00
Functional Requirements	Link
Technical Requirements	Link
Contracts	File: ./contracts/vestings/VestingBasic.sol SHA3: d1f90494c40be5d2faf77ee97d815aeae3182ef268720fffe22b3f7a8ee28983

Second review scope

Repository	https://github.com/gamesforaliving/web3-contracts-bundle
Commit	2813080762721cdb92e832b6a7df74974fe51fe3
Contracts Addresses	https://bscscan.com/address/0x6f4FC00457CAfb8fFF19715A6E38D6Fc386B6857#code https://bscscan.com/address/0x8EEDd042caCE47963F52E9D2190c97Ab00F33f03#code https://bscscan.com/address/0x08E6D346cef70D8F680D520256B75ba504cCB942#code https://bscscan.com/address/0xA46E64D618475ff8379f152A03317a108a89fDd1#code https://bscscan.com/address/0xb18771af81eFEd73911Bfe95389F0A28e946592d#code https://bscscan.com/address/0x5a1A22Dd99AF1835042572436Fb24b13b05f0375#code https://bscscan.com/address/0xD5dcB7b469eF01283B41EFE468593a71BEee48D6#code

	https://bscscan.com/address/0x39cf1cdF5152e851C769Ee4B8c7fFF489B1B36cF#code https://bscscan.com/address/0x5C3227A15a4917590f0Ea114652725D7030B5537#code
Functional Requirements	Link
Technical Requirements	Link
Contracts	File: ./contracts/vestings/VestingBasic.sol SHA3: 4d602ae1b549b44da5b6a7c35d99d9dbe6996bf21eeddc841e31e30b8a51ec4e

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 **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**.

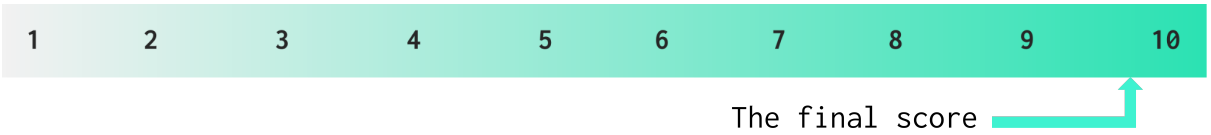

Table. The distribution of issues during the audit

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

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

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.

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

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.

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.	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 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.	Not Relevant
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155 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.	Not Relevant
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Not Relevant
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 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.	Not Relevant
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
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.	Failed
Stable Imports	**Custom**	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.)

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

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

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.