



**HACKEN**

# SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

**Customer:** Bictory Finance  
**Date:** November 11<sup>th</sup>, 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

<b>Name</b>	Smart Contract Code Review and Security Analysis Report for Bictory Finance
<b>Approved By</b>	Evgeniy Bezuglyi   SC Audits Department Head at Hacken OU
<b>Type</b>	SPL token; Staking
<b>Platform</b>	Solana
<b>Language</b>	Rust
<b>Methodology</b>	<a href="#">Link</a>
<b>Website</b>	<a href="https://bictory.io">https://bictory.io</a>
<b>Changelog</b>	01.11.2022 - Initial Review
	11.11.2022 - Second Review



## Table of contents

Introduction	4
Scope	4
Severity Definitions	6
Executive Summary	7
Checked Items	8
System Overview	11
Findings	12
Disclaimers	14

## Introduction

Hacken OÜ (Consultant) was contracted by Bictory Finance (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	<a href="https://gitlab.com/bictory/smartcontracts/bt-token/-/tree/bt-token/programs/vesting">https://gitlab.com/bictory/smartcontracts/bt-token/-/tree/bt-token/programs/vesting</a>
Commit	4ffc8d9818860fbee6e360593ae6b8ad7d2d73d2
Whitepaper	No
Functional Requirements	<a href="https://gitlab.com/bictory/smartcontracts/bt-token/-/blob/bt-token/programs/vesting/README.md">https://gitlab.com/bictory/smartcontracts/bt-token/-/blob/bt-token/programs/vesting/README.md</a>
Technical Requirements	<a href="https://gitlab.com/bictory/smartcontracts/bt-token/-/blob/bt-token/programs/vesting/README.md">https://gitlab.com/bictory/smartcontracts/bt-token/-/blob/bt-token/programs/vesting/README.md</a>
Contracts Addresses	<a href="https://solscan.io/token/GECuj9Vs2PkM59YF2hmiwgpwSzW7pxXBJsiK3ANGgVes">https://solscan.io/token/GECuj9Vs2PkM59YF2hmiwgpwSzW7pxXBJsiK3ANGgVes</a>
Contracts	<p>File: ./programs/vesting/src/processor/helper.rs            SHA3:            5571f8b2d5a20b92d20be9bc4d3eac394670f77a8a032f59a80cc9c4a42ca223</p> <p>File: ./programs/vesting/src/processor/mod.rs            SHA3:            b886fc762777aaa1dbc28d3ad53d4eca25c8041e5219420cfb0272aa2659af0e</p> <p>File: ./programs/vesting/src/processor/process_cancel.rs            SHA3:            301941ae8e6443818f464b9f55b1d561ddfa46165cc7eca27bd380baefd2e604</p> <p>File: ./programs/vesting/src/processor/process_claim.rs            SHA3:            b51bb708e579804d274d4cfe509c9bf4b46e55de94554b1f4801b5d731cbe97f</p> <p>File: ./programs/vesting/src/processor/process_initialize.rs            SHA3:            54347918617ede91bf02bc85da74be92d1f189dd0c0ca6410c2de8bd0f937046</p> <p>File: ./programs/vesting/src/processor/process_stake.rs            SHA3:            9975aca5fcdd0db52166c005f404a756fd2af682d944b7007f5a139fbc3e8c29</p> <p>File: ./programs/vesting/src/constants.rs            SHA3:            1dab66309fa8864560ec853c8e5157356bd37033291edf2d9ea64837d4bc6542</p> <p>File: ./programs/vesting/src/errors.rs</p>

	<p>SHA3: cd366487e05651a1cddd2ebca7516a27aa60de1aa85d922933d624d6e207d18f</p> <p>File: ./programs/vesting/src/lib.rs SHA3: b8f48dcb96fc9af10ec81fa249fa32b36d0ac564a57cd829ec7882aa7be4ffd3</p> <p>File: ./programs/vesting/src/states.rs SHA3: c2a6da157e4fd8dcae351923c112d9adef318bffd082059a35d91aaf75c7dd1</p> <p>File: ./programs/vesting/Cargo.toml SHA3: 3ca4b0460e92a1c32935dba91469b57346d7de30185d152957b3b7fe49c18989</p>
--	---

## Second review scope

<b>Commit</b>	f6a46174ac872a9fd05b66810219070f57cda238
<b>Contracts</b>	<p>File: ./programs/vesting/src/processor/helper.rs SHA3: 2d489693711c97bdd2acfa9bd35f0cb5d85f980cde8f7d0d3cc61d9025398419</p> <p>File: ./programs/vesting/src/processor/mod.rs SHA3: b886fc762777aaa1dbc28d3ad53d4eca25c8041e5219420cfb0272aa2659af0e</p> <p>File: ./programs/vesting/src/processor/process_cancel.rs SHA3: 5ecaf75444f1c8f07ae8b910d798a162fe71d477bd0980581b99920682271154</p> <p>File: ./programs/vesting/src/processor/process_claim.rs SHA3: 06da41af6dea7e01fce80fa7c5e8636d03dd202ffb35ef62fda964dee947fa2f</p> <p>File: ./programs/vesting/src/processor/process_initialize.rs SHA3: 6104ddf7f31d3ae3a6adcf70e311f01f2d3e5a9ce0cedf6222ca449d3d1cc3c1</p> <p>File: ./programs/vesting/src/processor/process_stake.rs SHA3: f33c27d9d1347ae8e6a122222aa4fbf465ff5cee3573d842867dfa8116ded02d</p> <p>File: ./programs/vesting/src/constants.rs SHA3: dd2befb4754452a36627143efac12a2a39e4c26fe45a8cd541bacd91e8bd5970</p> <p>File: ./programs/vesting/src/errors.rs SHA3: cd366487e05651a1cddd2ebca7516a27aa60de1aa85d922933d624d6e207d18f</p> <p>File: ./programs/vesting/src/lib.rs SHA3: 68c33657734e5e8eb036a414a5ea909cb70b60cad885d235c57e4c91427ee2f3</p>

	File: ./programs/vesting/src/states.rs SHA3: b10570f5a9f048b6f5ce44f80ead9cf0103f80c6394342d1212112f3572e1cf6  File: ./programs/vesting/Cargo.toml SHA3: 3ca4b0460e92a1c32935dba91469b57346d7de30185d152957b3b7fe49c18989
--	---

## Severity Definitions

Risk Level	Description
<b>Critical</b>	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
<b>High</b>	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.
<b>Medium</b>	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
<b>Low</b>	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 [scoring methodology](#).

### Documentation quality

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

- Functional requirements provided.
- Technical description is provided.

### Code quality

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

- Each part of the contract functionality is separated through files.
- The development environment is configured.

### Test coverage

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

- Basic user interactions are covered with tests.
- Negative cases coverage is missed.
- Interactions by several users are not tested thoroughly.

### Security score

As a result of the second audit, the code does not contain any 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**.



The final score



*Table. The distribution of issues during the audit*

Review date	Low	Medium	High	Critical
1 November 2022	2	0	1	0
11 November 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	Description	Status
Missing Signer Checks	Case when an instruction should only be available to a restricted set of entities, but the program does not verify that the call has been signed by the appropriate entity (e.g., by checking <code>AccountInfo::is_signer</code> ).	Passed
Missing Ownership Checks	For accounts that are not supposed to be fully user-controlled, the program does not check the <code>AccountInfo::owner</code> field.	Passed
Missing rent exemption checks	All Solana accounts holding an Account, Mint, or Multisig must contain enough SOL to be considered rent exempt. Otherwise the accounts may fail to load.	Passed
Signed invocation of unverified programs	The program does not verify the pubkey of any program called via the <code>invoke_signed()</code> API.	Passed
Solana account confusions	The program fails to ensure that the account data has the type it expects to have.	Passed
Redeployment with cross-instance confusion	The program fails to ensure that the wasm code has the code it expects to have	Passed
Arithmetic overflow/underflows	If an arithmetic operation results in a higher or lower value, the value will wrap around with two's complement.	Passed
Numerical precision errors	Numeric calculations on floating point can cause precision errors and those errors can accumulate.	Passed
Loss of precision in calculation	Numeric calculations on integer types such as division can loss precision.	Passed
Casting truncation	Potential truncation problem with a cast conversion	Not Relevant
Exponential complexity in calculation	Finding computational complexity in calculations.	Passed
Missing freeze authority checks	When freezing is enabled, but the program does not verify that the freezing account call has been	Not Relevant



	signed by the appropriate freeze_authority	
<b>Insufficient SPL-Token account verification</b>	Finding extra checks that should not exist with the given type of accounts	Passed
<b>Over/under payment of loans</b>	A loan overpayment is when your pay extra towards your loan over and above your agreed monthly repayment.  A loan underpayment is when your pay less towards your loan over and below your agreed monthly repayment	Passed
<b>Anti-pattern instruction calls</b>	Calling some anti-pattern instructions specific to Solana blockchain	Passed
<b>Unsafe Rust code</b>	The Rust type system does not check memory safety of unsafe Rust code. Thus, if a smart contract contains any unsafe Rust code, it may still suffer from memory corruptions such as buffer overflows, use after frees, uninitialized memory, etc.	Not Relevant
<b>Outdated dependencies</b>	Rust/Cargo makes it easy to manage dependencies, but the dependencies can be outdated or contain known security vulnerabilities. cargo-outdated can be used to check outdated dependencies.	Passed
<b>Redundant code</b>	Repeated code or dead code that can be cleaned or simplified to reduce code complexity.	Passed
<b>Do not follow security best practices</b>	Failing to properly use assertions, check user errors, multisig, and so on.	Passed
<b>Project specification implementation check</b>	Ensuring that the contract logic correctly implements the project specifications	Passed
<b>Contract-specific low-level vulnerabilities</b>	Examining the code in detail for contract-specific low-level vulnerabilities,	Passed
<b>Ruling out economic attacks</b>	Economic rules that can be exploited to steal funds	Passed
<b>DoS (Denial of Service)</b>	Execution of the code should never be blocked by a specific contract state unless it is required.	Passed

<b>Front-running or sandwiching</b>	Checking for instructions that allow front-running or sandwiching attacks	Passed
<b>Unsafe design vulnerabilities</b>	Checking for unsafe design which might lead to common vulnerabilities being introduced in the future	Passed
<b>As-of-yet solana unknown classes of vulnerabilities</b>	Checking for any other, as-of-yet unknown classes of vulnerabilities arising from the structure of the Solana blockchain	Passed
<b>Rug-pull mechanisms or hidden backdoors</b>	Checking for rug-pull mechanisms or hidden backdoors.	Passed

## System Overview

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

- *Token Bictory* – simple Solana SPL token that mints all initial supply to a deployer. Additional minting is not allowed.

It has the following attributes:

- Name: Bitcory
- Symbol: BT
- Decimals: 9
- Total supply: 100m tokens.
- *Vesting contract* – a contract that rewards users for staking their tokens. Users can claim their tokens according to the vesting scheduler. Vesting contract can be canceled anytime by the admin account, which has initialized it at the beginning, and all the remaining tokens will be transferred from Smart Contract to the actual holder of the tokens. In this contract, the duration of a month is considered 30 days, irrespective of the actual number of days in a month. For instance, if lock duration is 3 months, it will be 90 days from the date of TGE.

## Privileged roles

- The owner of the contract is able to cancel the ongoing vesting schedule by the admin's account. All the remaining tokens will be transferred from Smart Contract to the actual holder of the tokens.

## Risks

- In case of an admin keys leak, an attacker can cancel the contract, and all remaining tokens will be transferred from the Smart Contract to the actual holder of the tokens.

## Findings

### ■■■■ Critical

No critical severity issues were found.

### ■■■ High

#### 1. Incorrect Calculation for Next Pay Date

MONTHLY\_TIMESTAMP constant has an incorrect value **1** for testing purposes. `calc_next_pay_date` function uses MONTHLY\_TIMESTAMP to calculate the next pay date.

This can lead to a vesting schedule that allows claim tokens instantly without a cliff at the beginning every second after.

It depends on the 'release\_frequency' and 'cliff' values inside 'StreamInstruction' and 'StreamParams'.

**Files:** `./programs/vesting/src/constants.rs`

`./programs/vesting/src/processor/process_claim.rs`

`./programs/vesting/src/states.rs`

**Contract:** `process_claim`

**Function:** `calc_next_pay_date`

**Recommendation:** Remove commented code.

```
// pub const MONTHLY_TIMESTAMP: u64 = 60 * 60 * 24 * 30; // Assume 30
days per month
```

or remove test value settings for MONTHLY\_TIMESTAMP

```
pub const MONTHLY_TIMESTAMP: i64 = 1; // Set test value as 1
```

**Status:** **Fixed** (Revised commit: f6a4617)

### ■■ Medium

No medium severity issues were found.

### ■ Low

#### 1. Unused Variable

GloabalState field 'admin' is never used. Pubkey 'admin' is supposed to be used for validation during the 'cancel' instruction.

**File:** `./programs/vesting/src/states.rs`

**Instruction:** `cancel`

**Recommendation:** Remove unused variable or use 'admin' field for validation during 'cancel' instruction.

**Status:** **Fixed** (Revised commit: f6a4617)

[www.hacken.io](http://www.hacken.io)

## 2. Commented Code Parts

Commented parts of code in a contract. They will not cause any security issues but will make the code less clear.

In the file *constants.rs*, line 4 are commented parts of code.

This reduces code quality.

**File:** `./programs/vesting/src/constants.rs`

**Constant:** `MONTHLY_TIMESTAMP`

**Recommendation:** Remove commented parts of code.

**Status:** `Fixed` (Revised commit: `f6a4617`)

## 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.