HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Dexalot
Date: October 11th, 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 Dexalot
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	ERC-20 vesting and asset management
Platform	EVM
Network	Ethereum
Language	Solidity
Methods	Manual Review, Automated Review, Architecture Review
Website	https://dexalot.com/
Timeline	20.07.2022 - 11.10.2022
Changelog	02.08.2022 - Initial Review 25.08.2022 - Second Review 12.09.2022 - Third Review 27.09.2022 - Fourth Review 11.10.2022 - Fifth 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 Dexalot (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/Dexalot/contracts/
Commit:
86a5b953a3886b9e176a7c75cc4054cc70b4fa89
Technical Documentation:
Type: Whitepaper (partial functional requirements provided)
Link

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/interfaces/IPortfolio.sol
SHA3: 72ad0a4c55026222f480dd59517f5cd837101d9d08e5d90db1fbeed93e1173f7

File: ./contracts/interfaces/ITradePairs.sol
SHA3: 57b640c8923728d4a3e0577dc61683287030c4c81bd2165b25f010557584a5bd

File: ./contracts/token/TokenVesting.sol
SHA3: cc3a08e4c946d06119a15fd6ab57ce35dfae5b93454462600ba75cff69cc97ce

File: ./contracts/token/TokenVestingCloneable.sol
SHA3: ca07fe83bdfd2dbf257ed595a8412fabaef4e7ad9895fe30d026fd5334b77386

File: ./contracts/token/TokenVestingCloneFactory.sol
SHA3: 19eeb4e638d482743df5adc51d04cd088fd7e4b9e457f4fa2fba26ad6477f977

File: ./contracts/token/TokenVestingV1.sol
SHA3: 92a0b29af2573e00f2ea0c89e2daca91456d27aa661d7e3c300479d81901a195

File: ./contracts/library/StringLibrary.sol
SHA3: 48d0bb010ed81c424ccb28d1f16d15a1dd53cab608a8667e4dad196427e5549f

Second review scope

Repository:
https://github.com/Dexalot/contracts/
Commit:
18c340f436a235b9504303268fafc8be9940ed97
Technical Documentation:
Type: Whitepaper (partial functional requirements provided)
Link

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/interfaces/IPortfolio.sol
SHA3: 2d868597482d62ebe4e6f84cd7fd62d9ee7d39a959a27ad9992d61e38ad8ab8a

File: ./contracts/interfaces/ITradePairs.sol
SHA3: 57b640c8923728d4a3e0577dc61683287030c4c81bd2165b25f010557584a5bd

File: ./contracts/token/TokenVesting.sol
SHA3: d45711fb70d032596d17ffab187b9d5b821f2557cc43035cdc600f337d1cdd21

File: ./contracts/token/TokenVestingCloneable.sol
SHA3: c05cfa0fb6ce82265593143aaad8fb840d8dea208f759ec828201d37da0edd56

File: ./contracts/token/TokenVestingCloneFactory.sol
SHA3: 69c5d42d0f99e0ec7f27eda95a791ae0f284091e5f21d1430d303fc9f72b9250

File: ./contracts/token/TokenVestingV1.sol
SHA3: 1c4bb1bdad092fa2786cfb036848fed771c5fa8e43b755d7eb904a0f9205851b

File: ./contracts/library/StringLibrary.sol
SHA3: 48d0bb010ed81c424ccb28d1f16d15a1dd53cab608a8667e4dad196427e5549f

Third review scope

Repository:
https://github.com/Dexalot/contracts/
Commit:
c2746740797c841c1166c39c6e30c64e5e1baf2b
Technical Documentation:
Type: Whitepaper (partial functional requirements provided)
Link

Integration and Unit Tests: Yes
Contracts:

File: ./contracts/interfaces/IPortfolio.sol
SHA3: 2d868597482d62ebe4e6f84cd7fd62d9ee7d39a959a27ad9992d61e38ad8ab8a

File: ./contracts/interfaces/ITradePairs.sol
SHA3: b0dcbe59c7cf364ad2102d9322e0de998aa4b798fa936b85c89332bb8d477d8f

File: ./contracts/library/StringLibrary.sol
SHA3: 48d0bb010ed81c424ccb28d1f16d15a1dd53cab608a8667e4dad196427e5549f

File: ./contracts/token/TokenVesting.sol
SHA3: 56fe41a95a23ae46bf77c08df30f44d6a58b127891928980bc2056194d92e225

File: ./contracts/token/TokenVestingCloneable.sol
SHA3: 01d24e1046205b28401c803675b7d43dfb355f9ccd8a624e29476651ebf59937

File: ./contracts/token/TokenVestingCloneFactory.sol
SHA3: 9aa30fc19b5b1ef4fb553ddf6bb812f9c5a5cfeb4c101acedfbb30f7b00a0bc0

File: ./contracts/token/TokenVestingV1.sol
SHA3: 80e0e721cfb9f6db5fa4235f8327069b94a883f94ec106b3ca0ac558945eb841

Fourth review scope

Repository:
https://github.com/Dexalot/contracts/
Commit:
ba38d6e804e49fa58e7cabc6a677218f1f72a5d0
Technical Documentation:
Type: Whitepaper (partial functional requirements provided)
Link

Integration and Unit Tests: Yes
Contracts:

File: ./contracts/interfaces/IPortfolio.sol
SHA3: 62121d2234b3217a42e718d61bf198cc53d0ab35ac36e5880f263f2c701c7da2

File: ./contracts/interfaces/ITradePairs.sol
SHA3: 27e302ba30023cf81228c674743ed963036274df8fe7a96e525e3a362f6a478e

File: ./contracts/library/UtilsLibrary.sol
SHA3: 3bdd91968ee2ee3b9f9486cd9c733c0060c3f82c506f9ff67244d65607b3649f

File: ./contracts/token/TokenVestingCloneable.sol
SHA3: 65840a65289ce9d966b322411380f88ef7271c36a0ffeb3fd5a1796818904e75

File: ./contracts/token/TokenVestingCloneFactory.sol
SHA3: 6398e269dce1e18a54b250b732eb11a2bce679c756b68e09d6ef4af1b9e1e2a7

Fifth review scope

Repository:
https://github.com/Dexalot/contracts/
Commit:
5dbc686c9fd225eba52f9390270a0f5b8aa56262
Technical Documentation:
Type: Litepaper and technical description
Link
Integration and Unit Tests: Yes
Contracts:

File: ./contracts/interfaces/IPortfolio.sol
SHA3: f20f67f62f6e31ff36c9f58301083bc97da89db1036cd0e79940724e662daf02

File: ./contracts/interfaces/ITradePairs.sol
SHA3: 9e2b3e3b87307cce57170c6f5f7837e05ce70a63dbab9bfba34b9569985250b1

File: ./contracts/library/UtilsLibrary.sol
SHA3: 3bdd91968ee2ee3b9f9486cd9c733c0060c3f82c506f9ff67244d65607b3649f

File: ./contracts/token/TokenVestingCloneable.sol
SHA3: 536b631ac0cfc606a11881f22bb17f0d80d633c01486ffae40797f5942b039d6

File: ./contracts/token/TokenVestingCloneFactory.sol
SHA3: cb55125974cf54b67b7b0158dcf8f375e50fe3c2faa58417f510c7e7251ffec4

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 total Documentation Quality score is **10** out of **10**. Functional requirements were not provided. A technical description is provided as comments in the code.

Code quality

The total CodeQuality score is **10** out of **10**. Most of the code follows official language style guides. Unit tests were provided.

Architecture quality

The architecture quality score is **10** out of **10**. The architecture is clear. Development environment was provided.

Security score

As a result of the audit, the code contains **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**.

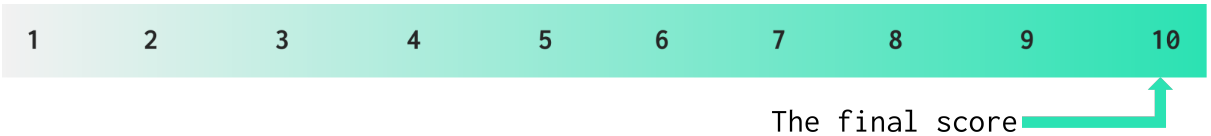

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
02 August 2022	3	2	1	1
19 August 2022	3	0	0	0
07 September 2022	1	0	0	0
26 September 2022	0	0	0	0
04 October 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	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.	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.	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	SWC-115	tx.origin should not be used for	Passed

through tx.origin		authorization.	
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 EIP-155	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifier should always be used. All parameters from the signature should be used in signer recovery	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 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.	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	Passed

		data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	
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 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

System Overview

Vesting is an ERC-20 vesting system with the following contracts:

- *TokenVestingCloneable* – is a contract for ERC-20 tokens linear vesting with the “1.0.3” version. Tokens are distributed gradually after the cliff period until the vesting end time. Tokens are transferred to the beneficiary address. The contract can have a period value: vesting amount is gradually grown in each period (for example, every 30 days, the vesting amount is updated).

  The defined percentage of tokens is vested before the vesting starts to the beneficiary and then transferred to the *Portfolio* contract, it is available after the defined time. If these tokens were not vested in the correct period, they would be transferred in vesting to the beneficiary address.

  There is a functionality accessible for the owner that allows to revoke the vesting for each token (funds not vested yet are transferred to the owner).

  The contract allows the vesting of different ERC-20 tokens, but all the parameters for vesting are the same.
- *TokenVestingCloneFactory* – is a factory contract with the "1.0.0" version that allows the creation of *TokenVestingCloneable* contracts.
- *UtilsLibrary* – is a library with utility functions, used in *TokenVestingCloneable* contracts.
- *IPortfolio* – is an interface for the *Portfolio* contract (out of scope), used in *TokenVesting*, *TokenVestingCloneable*, *TokenVestingV1* contracts.
- *ITradePairs* – is an interface for the *TradePair* contract (out of scope), used in the *IPortfolio* contract.

Privileged roles

- The owner of the *TokenVestingCloneable* contract can update *Portfolio* contract address, revoke vestings.
- The owner of the TokenVestingCloneFactory can create *TokenVestingCloneable* contracts.

Risks

- There are contracts in the repository not included in the audit scope. They cannot be verified.
- The tokens to be vested should be transferred to vesting contracts only before token payments for the correct calculations.
- *TokenVestingCloneable* contract interacts with the *Portfolio* contract, which is out of scope; its secureness can not be guaranteed.

Findings

■■■■ Critical

1. Requirements Violation

In the *releaseToPortfolio* function, an *unreleased* amount of tokens are transferred to the *_beneficiary* address. Then these funds are transferred from *_beneficiary* address to the *Portfolio* contract in the *Portfolio.depositTokenFromContract* function using the *safeTransferFrom* function without their prior approval.

Therefore, it is impossible to release tokens to the *Portfolio* contract.

Files: ./contracts/Portfolio.sol, ./contracts/token/TokenVesting.sol, ./contracts/token/TokenVestingCloneable.sol, ./contracts/token/TokenVestingV1.sol

Contracts: Portfolio, TokenVesting, TokenVestingCloneable, TokenVestingV1

Functions: Portfolio.depositTokenFromContract, TokenVesting.releaseToPortfolio, TokenVestingCloneable.releaseToPortfolio, TokenVestingV1.releaseToPortfolio

Recommendation: Approve an *unreleased* amount of tokens of the *_beneficiary* address to the *Portfolio* contract before transferring to the Portfolio contract (before calling the *depositTokenFromContract* function).

Status: Mitigated (The Customer comment: “User must give two approvals for the vesting and portfolio contracts before calling this function.”)

■■■ High

1. Data Consistency

The functionality allows the owner to update the percentages for the amount vested at TGE (*setPercentage* function) and to reinstate the vesting (*reinstate*).

Therefore, the *_vestedByPercentage* function may incorrectly calculate the amount of tokens released at TGE (*releaseToPortfolio*) if the percentage value is changed after the releasing tokens at TGE; if after the vesting reinstatement, there is a different total amount of the tokens (current + released) than it was when the releasing tokens at TGE.

Due to this, the vested amount will be calculated incorrectly in the *_releasableAmount* function, as the *_vestedByPercentage(token)* value will not be equal to the actual paid part of

_released[address(token)]. The *canFundWallet* function will incorrectly indicate if the vesting has been funded to the Portfolio.

Files: ./contracts/token/TokenVesting.sol, ./contracts/token/TokenVestingCloneable.sol, ./contracts/token/TokenVestingV1.sol

Contracts: TokenVesting, TokenVestingCloneable, TokenVestingV1

Functions: setPercentage, reinstate

Recommendation: Ensure that percentages cannot be changed after the releasing tokens at TGE (*releaseToPortfolio*) and that the balance after the reinstatement is appropriate.

Status: Fixed (Revised commit: 18c340f436a235b9504303268fafc8be9940ed97)

Medium

1. Redundant Functionality

stringToBytes32 function is redundant as *stringToBytes32* from the *StringLibrary* can be used directly.

Files: ./contracts/token/TokenVesting.sol, ./contracts/token/TokenVestingCloneable.sol, ./contracts/token/TokenVestingV1.sol

Contracts: TokenVesting, TokenVestingCloneable, TokenVestingV1

Function: stringToBytes32

Recommendation: Remove the redundant function, use *stringToBytes32* from the *StringLibrary* directly.

Status: Fixed (Revised commit: 18c340f436a235b9504303268fafc8be9940ed97)

2. Missing Events Emit on Changing Important Values

The contract does not emit any events after changing important values.

Files: ./contracts/token/TokenVesting.sol, ./contracts/token/TokenVestingCloneable.sol, ./contracts/token/TokenVestingV1.sol, ./contracts/Portfolio.sol

Contracts: TokenVesting, TokenVestingCloneable, TokenVestingV1, Portfolio

Functions: TokenVesting.setPercentage, TokenVesting.setStartPortfolioDeposits, TokenVesting.setPortfolio, TokenVestingCloneable.setPercentage, TokenVestingCloneable.setStartPortfolioDeposits, TokenVestingCloneable.setPortfolio, TokenVestingV1.setPercentage, TokenVestingV1.setStartPortfolioDeposits, TokenVestingV1.setPortfolio, Portfolio.setNative

Recommendation: Implement event emits after changing the contract values.

Status: Fixed (Revised commit: 18c340f436a235b9504303268fafc8be9940ed97)

Low

1. Floating Pragma

The project`s contracts use floating pragma ^0.8.4.

Contracts with unlocked pragmas may be deployed by the latest compiler, which may have higher risks of undiscovered bugs. Contracts should be deployed with the same compiler version they have been tested thoroughly.

Files: ./contracts/token/TokenVesting.sol, ./contracts/token/TokenVestingCloneable.sol, ./contracts/token/TokenVestingV1.sol, ./contracts/token/TokenVestingCloneFactory.sol, ./contracts/library/StringLibrary.sol, ./contracts/interfaces/IPortfolio.sol, ./contracts/interfaces/ITradePairs.sol

Contracts: TokenVesting, TokenVestingCloneable, TokenVestingV1, TokenVestingCloneFactory, StringLibrary, IPortfolio, ITradePairs

Recommendation: Consider locking the pragma version whenever possible.

Status: Fixed (Revised commit: ba38d6e804e49fa58e7cabc6a677218f1f72a5d0)

2. Functions that Can Be Declared External

There are public functions in the contracts that are not used internally.

“External” visibility uses less Gas.

Files: ./contracts/token/TokenVesting.sol, ./contracts/token/TokenVestingCloneable.sol, ./contracts/token/TokenVestingV1.sol

Contracts: TokenVesting, TokenVestingCloneable, TokenVestingV1

Functions: TokenVesting.canFundWallet, TokenVestingCloneable.canFundWallet, TokenVestingV1.canFundWallet

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

Status: Fixed (Revised commit: c2746740797c841c1166c39c6e30c64e5e1baf2b)

3. Block Values as a Proxy for Time Using

The contract uses *block.timestamp* for time calculations. It is not precise and safe.

Files: ./contracts/token/TokenVesting.sol, ./contracts/token/TokenVestingCloneable.sol, ./contracts/token/TokenVestingV1.sol

Contracts: TokenVesting, TokenVestingCloneable, TokenVestingV1

Functions: constructor, canFundWallet, canFundPortfolio, release, _vestedAmount, _vestedByPercentage

Recommendation: It is recommended to avoid using *block.timestamp* in the time calculations. Alternatively, it is safe to use oracles.

Status: Mitigated (The Customer notice: “ This vesting contract depends on time-based vesting schedule using block timestamps. Therefore, the contract would be susceptible to timestamp manipulation miners may be able to do in some EVMs for variables with less than a min time lengths for delta time. To mitigate potential exploits variables holding delta time are required to be more than 5 minutes.”)

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.