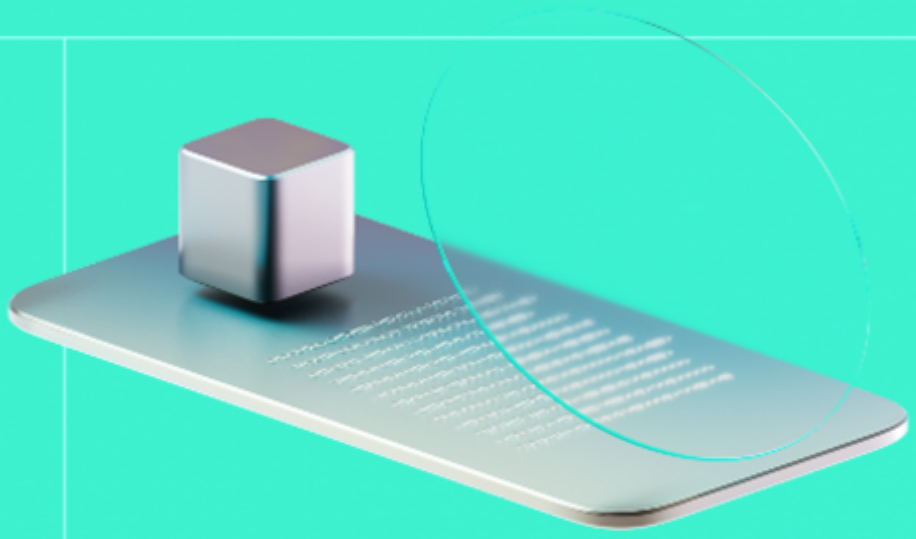
HACKEN

Smart Contract Code Review And Security Analysis Report

Customer: Farcana

Date: 29/12/2023



We thank Farcana for allowing us to conduct a Smart Contract Security Assessment. This document outlines our methodology, limitations, and results of the security assessment.

Farcana integrates blockchain technology with traditional game development for a seamless, immersive gaming experience.

Platform: EVM

Language: Solidity

Tags: ERC-20, Vesting

Timeline: 12/12/2023 - 29/12/2023

Methodology: https://hackenio.cc/sc_methodology

Last Review Scope

Repository	https://github.com/farcana/smart-contracts
Commit	3a8a187

Audit Summary

10/10	9/10	0%	6/10
Security Score	Code quality score	Test coverage	Documentation quality score

Total 9.4/10

The system users should acknowledge all the risks summed up in the risks section of the report

7	7	0	0
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	0
High	0
Medium	2
Low	3

Vulnerability	Status
F-2023-0124 - Insufficient Parameter Validation in addVestingSchedule() Function of Vesting Contract	Fixed
F-2023-0125 - Unrestricted Authority to Alter Vesting Beneficiary Addresses	Fixed
F-2023-0126 - Single-Step Ownership Transfer	Fixed
F-2023-0127 - Inappropriate Mutability of tgeTime in Vesting Contract	Fixed
F-2023-0128 - Non-Utilization of SafeERC20 for Token Transfers in Vesting Contract	Fixed
F-2023-0129 - Outdated Compiler Version	Fixed
F-2023-0135 - Missing zero address checks	Fixed

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

Name	Smart Contract Code Review and Security Analysis Report for Farcana
Audited By	Ivan Bondar
Approved By	Przemyslaw Swiatowiec, Ataberk Yavuzer
Website	https://www.farcana.com
Changelog	29/12/2023 - Final Report

Table to Contents

System Overview

Farcana integrates blockchain technology with traditional game development for a seamless, immersive gaming experience. As the first project of its kind, our flagship multiplayer online game, Farcana, leverages web3 technologies. The game has a fully player-controlled economy, marketplace, and asset ownership.

The files in the scope:

- FarcanaToken — is an ERC20 token. Upon deployment, it is initialized with a specified name, symbol, decimal count, total supply, and an owner address. The token supply is predetermined and fixed, with no provisions for minting additional tokens post-deployment. However, it includes a burn function, allowing the owner to reduce the total supply by burning tokens.
- FarcanaVesting — token vesting mechanism designed for the Farcana ecosystem. It allows for the creation of multiple vesting schedules per beneficiary, enabling the controlled and gradual release of tokens. Key features include:
  - Flexible Vesting Schedules: Each beneficiary can have multiple vesting schedules, each with its own cliff, duration, start time, and total amount of tokens to be released.
  - Claim Functionality: Beneficiaries can claim their vested tokens once they are due for release.
  - Substantial Initial Release: The contract is designed such that a significant portion of tokens becomes claimable immediately after the cliff period, with the remaining amount vested linearly over the rest of the duration.

Privileged roles

- FarcanaToken.sol:
  - Owner:
    - Capable of burning tokens from their own holdings.
    - Authorized to transfer ownership of the contract.
- FarcanaVesting.sol:
  - Owner:
    - Authorized to set the TGE (Token Generation Event) timestamp.
    - Empowered to add vesting schedules for any beneficiary.
    - Able to change the beneficiary address for existing vesting schedules.
    - Can access and review all data related to a specific beneficiary's vesting schedule.

Executive Summary

This report presents an in-depth analysis and scoring of the Customer's smart contract project. Detailed scoring criteria can be referenced in the scoring methodology.

Documentation quality

The total Documentation Quality score is **6** out of **10**.

- Functional requirements are not provided.
  - Overall system requirements are provided.
  - No roles description.
  - No Tokenomics.
- Technical description is not provided.
  - NatSpec is partially missing.

Code quality

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

- The development environment is not configured.

Test coverage

Code coverage of the project is 0.0% (branch coverage).

- No tests.

Security score

Upon auditing, the code was found to contain **0** critical, **0** high, **2** medium, and **3** low severity issues, leading to a security score of **10** out of **10**.

All identified issues are detailed in the “Findings” section of this report.

Summary

The comprehensive audit of the Customer's smart contract yields an overall score of **9.4**. This score reflects the combined evaluation of documentation, code quality, test coverage, and security aspects of the project.

Risks

- Beneficiary Change Risk:
  - The FarcanaVesting contract grants the owner the authority to modify the beneficiary address of any existing vesting schedule. While this offers flexibility, it also introduces a significant risk factor. If the owner mistakenly inputs an incorrect address or in the event of a compromise of the owner's keys, the tokens intended for one beneficiary could be erroneously or maliciously redirected to another address.

Findings

Vulnerability Details

F-2023-0124 - Insufficient Parameter Validation in addVestingSchedule() Function of Vesting Contract - Medium

Description:

The `addVestingSchedule` function in the TokenVesting contract is responsible for creating new vesting schedules for `beneficiaries`. However, the function does not perform adequate checks on the input parameters such as `cliff`, `duration`, `start`, and their relationship to each other and to `tgeTime`.

Improper parameter validation can result in:

- Immediate availability of tokens, bypassing the intended vesting schedule.
- Tokens being locked indefinitely if the vesting period is set too far in the future.
- Inaccurate vesting timelines that do not align with the project's intended token distribution plan.

Additionally, the contract does not have a mechanism to ensure that the total amount of tokens set for vesting is actually available or reserved in the contract, potentially leading to situations where the contract cannot fulfill the vesting due to insufficient tokens.

Assets:

- FarcanaVesting.sol [https://github.com/farcana/smart-contracts]

Status: Fixed

Classification

Severity: Medium

Impact: 4/5

Likelihood: 1/5

Recommendations

Recommendation:

- Implement Parameter Validations:
  - Ensure `start` is greater than or equal to `tgeTime`.
  - Validate that `cliff` and `duration` are positive and logically consistent (e.g., `cliff` should not exceed `duration`).

    - Check that `start` + `duration` does not result in an impractical vesting end time.
- Vesting Token Availability Check: Introduce a mechanism to verify that the contract holds enough tokens to cover the total amount set for all vesting schedules. This could be a check against the contract's token balance.

Remediation (Revised commit: 3a8a187) : The proposed recommendations for enhancing the parameter validations in the vesting contract have been successfully implemented.

F-2023-0125 - Unrestricted Authority to Alter Vesting Beneficiary Addresses - Medium

Description: The TokenVesting contract includes a function `changeBeneficiaryAddress`, which allows the contract owner to alter the `beneficiary` address of any vesting schedule. This function grants the owner unchecked power to transfer the claim rights of vested tokens from one address to another at any point, without any constraints or oversight mechanisms.

The ability to unilaterally change `beneficiary` addresses could lead to potential misuse or abuse of power, jeopardizing the security and fairness of the vesting process.

Assets:

- FarcanaVesting.sol [https://github.com/farcana/smart-contracts]

Status: Fixed

Classification

Severity: Medium

Impact: 4/5

Likelihood: 1/5

Recommendations

Recommendation: It is recommended to:

- Introduce Checks and Balances: Implement restrictions or multi-signature requirements for changing beneficiary addresses. This could include a time delay for changes to take effect, notification mechanisms, or requiring additional approvals.
- Document Function Purpose and Risks: Clearly document the intended use case for this function, outlining potential risks and the rationale behind allowing such a significant level of control.
- Consider Removing or Restricting Function: Evaluate the necessity of this function. If it s not critical to the vesting process, consider removing it. If it must remain, introduce stringent conditions under which it can be used, such as in the case of a proven lost or compromised original beneficiary address.

Remediation (Revised commit: 3a8a187) : The issue highlighting the potential misuse of this function due to its unrestricted power has been addressed through enhanced documentation. While the function itself remains

unchanged, NatSpec comments have been added to clearly articulate its purpose and use case.

The updated NatSpec comments clarify that this function is specifically designed for scenarios where a beneficiary loses access to their wallet. It ensures that while the beneficiary's address can be replaced, there is no minting of additional tokens or premature unlocking of existing tokens.

F-2023-0126 - Single-Step Ownership Transfer - Low

Description:

The current implementation of the FarcanaToken and FarcanaVesting contract utilizes a single-step process for ownership transfer.

This approach, while straightforward, does not include a verification step for the new owner address before finalizing the transfer. The absence of such a precautionary measure can lead to significant security and operational risks, particularly if an incorrect address is provided during the ownership transfer process. Mistakes or malicious activities could result in the permanent transfer of ownership to an unintended address, potentially leading to loss of control over the contract's administrative functionalities.

- Security Risks:
  - The single-step ownership transfer process increases the risk of accidental or malicious transfers, as there is no opportunity to verify or cancel the transfer once initiated.
- Operational Risks:
  - An incorrect transfer of ownership could result in administrative functions becoming inaccessible, potentially crippling the contract's operations and management.

Assets:

- FarcanaVesting.sol [https://github.com/farcana/smart-contracts]
- FarcanaToken.sol [https://github.com/farcana/smart-contracts]

Status: Fixed

Classification

Severity: Low

Impact: 1/5

Likelihood: 1/5

Recommendations

Recommendation:

Implement the Ownable2Step extension or a similar mechanism that introduces a two-step process for ownership transfer. This process typically involves nominating a new owner and then requiring a separate confirmation step to finalize the transfer.

Remediation (Revised commit: 3a8a187) : The issue concerning the single-step ownership transfer process in the FarcanaToken and FarcanaVesting contracts has been effectively addressed by implementing the Ownable2Step extension in both contracts.

F-2023-0127 - Inappropriate Mutability of tgeTime in Vesting Contract - Low

Description:

The vesting contract initializes a `tgeTime` variable in its `constructor` and provides a setter function `setTGEtime` that allows the owner to modify this time. Given that `tgeTime` represents the Token Generation Event time, a fundamental contract parameter, its ability to be altered post-deployment poses a risk of misuse or errors. Although currently unused in calculations, if `tgeTime` is intended for future critical functionalities, its mutable nature could compromise the contract's integrity.

Allowing the TGE time to be changed can lead to uncertainty and potential misuse. It undermines the reliability of the contract, especially if `tgeTime` becomes integral to vesting calculations or other operations in future iterations or integrations.

Assets:

- FarcanaVesting.sol [https://github.com/farcana/smart-contracts]

Status: Fixed

Classification

Severity: Low

Impact: 1/5

Likelihood: 1/5

Recommendations

Recommendation:

It is recommended to:

- Remove Initial Setting in Constructor: Eliminate the initialization of `tgeTime` in the contract's `constructor`. This will prevent the setting of `tgeTime` at the time of contract deployment.
- Single Setting in setTGEtime: Modify the `setTGEtime` function to allow the `tgeTime` to be set only once. This can be achieved by checking if `tgeTime` is still at its default value (e.g., 0) before allowing it to be set.
- Ensure Immutability Post-Setting: Once `tgeTime` is set through the `setTGEtime` function, it should become immutable, preventing any further changes. This enforces the integrity and constancy of the `tgeTime` after its initial definition.

Remediation (Revised commit:) : The previously identified issue regarding the mutability of the tgeTime variable in the vesting contract has been successfully addressed.

- The tgeTime initialization within the contract's constructor has been eliminated.
- The setTGEtime function has been modified to ensure that tgeTime can be set only once.

F-2023-0128 - Non-Utilization of SafeERC20 for Token Transfers in Vesting Contract - Low

Description:

The TokenVesting contract correctly imports and declares the use of OpenZeppelin's SafeERC20 library for safer ERC20 token interactions. However, the implementation inconsistently applies these safety measures. Specifically, the `claim` function executes a token transfer using the standard `transfer` method of the ERC20 token, bypassing the safety checks provided by SafeERC20.

Affected Code:

```
function claim() external {
    uint256 unreleased = prepareAvailableTokensForRelease(msg.sender);
    require(unreleased > 0, "No tokens are due for release");
    token.transfer(msg.sender, unreleased);
}
```

Using the standard `transfer` method without the safety checks of SafeERC20 can lead to unhandled exceptions, especially if the token contract does not return a boolean value as per the ERC20 standard. This inconsistency could result in failed or stuck transactions without proper error handling, potentially affecting the reliability of the vesting process.

Assets:

- FarcanaVesting.sol [https://github.com/farcana/smart-contracts]

Status: Fixed

Classification

Severity: Low

Impact: 1/5

Likelihood: 1/5

Recommendations

Recommendation:

It is recommended to refactor the `token.transfer(msg.sender, unreleased)` call in the `claim` function to use SafeERC20's safe transfer method: `token.safeTransfer(msg.sender, unreleased)`.

Remediation (Revised commit: 3a8a187) : The contract was updated to replace the standard `transfer` method with `safeTransfer` from the SafeERC20 library.

F-2023-0129 - Outdated Compiler Version - Info

Description: The FarcanaToken is written in Solidity version 0.4.24, which is significantly outdated. Using older versions of Solidity exposes the contract to potential security vulnerabilities that have been addressed in later versions. Additionally, newer versions of Solidity offer improved language features, optimizations, and security enhancements that are not present in earlier versions.

The use of an outdated version could lead to reduced efficiency, and compatibility issues with newer tools and libraries. It also prevents the adoption of current best practices in smart contract development.

Assets:

- FarcanaToken.sol [https://github.com/farcana/smart-contracts]

Status: Fixed

Classification

Severity: Info

Recommendations

Recommendation: Upgrade version to the up-to-date compiler version. This will provide access to the latest language features, security improvements, and optimizations.

Remediation (Revised commit: 3a8a187) : The FarcanaToken contract has been updated to use Solidity version 0.8.20.

F-2023-0135 - Missing zero address checks - Info

Description: The TokenVesting contract's `constructor` and `addVestingSchedule` function lacks validation checks for zero addresses. Specifically:

- In the `constructor`, the `token_address` parameter is assigned to the `token` variable without validating whether it is a non-zero address. This oversight could lead to the contract being initialized with an invalid token address.
- The `addVestingSchedule` function does not validate if the beneficiary address is a non-zero address. Adding a vesting schedule for a zero address could result in tokens being vested to an inaccessible address.

Assets:

- FarcanaVesting.sol [https://github.com/farcana/smart-contracts]

Status: Fixed

Classification

Severity: Info

Recommendations

Recommendation: Implement following address validations:

- In the `constructor`, add a check to ensure that `token_address` is not the zero address.
- In `addVestingSchedule`, introduce a similar check for the beneficiary parameter to prevent adding vesting schedules for the zero address.

Remediation (Revised commit: 3a8a187) : Checks to ensure that addresses are not zero was successfully implemented.

Observation Details

F-2023-0130 - Copy Of Well Known Contract - Info

Description: The current implementation of the token contract, while clear and adhering to the ERC20 standard, has opportunities for enhanced clarity and efficiency. The OpenZeppelin library provides a well-tested and secure foundation for ERC20 tokens, including functionalities that are currently manually implemented in the contract.

Reimplementing standard functionalities can introduce unnecessary complexity and potential for errors. Leveraging a widely-used and audited codebase like OpenZeppelin's ERC20 implementation can enhance security and reduce the likelihood of bugs.

Assets:

- FarcanaToken.sol [https://github.com/farcana/smart-contracts]

Status: Fixed

Recommendations

Recommendation: It is recommended to:

- Adopt OpenZeppelin's ERC20 Base: Use OpenZeppelin's ERC20 implementation as the foundational framework for token contract.
  - Custom Functionality: Add specific functionalities, such as the `burn` method.
  - Override Decimals: If necessary, override the `decimals` function for custom decimal values different from the standard 18.
- Direct Source Import:
  - Ensure Latest Updates: Directly import contracts from OpenZeppelin's source repositories to access the latest updates and security enhancements.
  - Version Pinning: Specify the exact version of OpenZeppelin contracts to use, ensuring compatibility and preventing issues from future changes.

Remediation (Revised commit: 3a8a187) : The FarcanaToken contract was effectively revised to incorporate OpenZeppelin's ERC20 base along with the Ownable2Step extension.

F-2023-0131 - Floating Pragma - Info

Description:

A **floating pragma** in Solidity refers to the practice of using a pragma statement that does not specify a fixed compiler version but instead allows the contract to be compiled with any compatible compiler version. This issue arises when pragma statements like pragma solidity ^0.8.0 are used without a specific version number, allowing the contract to be compiled with the latest available compiler version. This can lead to various compatibility and stability issues.

Version Compatibility: Using a floating pragma makes the contract susceptible to potential breaking changes or unexpected behavior introduced in newer compiler versions. Contracts that rely on specific compiler features or behaviors may break when compiled with a different version.

Interoperability Issues: Contracts compiled with different compiler versions may have compatibility issues when interacting with each other or with external services. This can hinder the interoperability of the contract within the Ethereum ecosystem.

The project uses floating pragmas ^0.8.0 and ^0.4.24.

Assets:

- FarcanaToken.sol [https://github.com/farcana/smart-contracts]
- FarcanaVesting.sol [https://github.com/farcana/smart-contracts]

Status: Fixed

Recommendations

Recommendation:

Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment. Consider known bugs for the compiler version that is chosen.

Remediation (Revised commit: 3a8a187) : The floating pragma issue was addressed by locking the pragma to a Solidity version 0.8.20.

F-2023-0132 - Redundant SafeMath - Info

Description:

Prior to Solidity version 0.8.0, arithmetic overflows were not handled natively by the language, and developers were encouraged to use the SafeMath library as a safeguard against such errors.

However, with the release of Solidity version 0.8.0, the language introduced new arithmetic overflow and underflow protection features that made the SafeMath library redundant if using Solc versions above 0.8.0.

Assets:

- FarcanaVesting.sol [https://github.com/farcana/smart-contracts]

Status: Fixed

Recommendations

Recommendation:

- For Vesting Contract:
  - Remove SafeMath : vesting contract utilizes Solidity version 0.8.0 or higher, the use of the SafeMath library becomes redundant.
  - Leverage Native Features: Rely on Solidity's native arithmetic overflow and underflow protection, which are integral features from version 0.8.0 onwards.
- For Token Contract (If Upgraded from Solidity version 0.4.24):
  - Apply Same Practices as Vesting Contract: In the event of upgrading the Solidity version of the token contract to 0.8.0 or above, similar recommendations apply.
  - Remove SafeMath Dependency: Utilize Solidity's built-in arithmetic protections instead of the SafeMath library.

Remediation (Revised commit: 3a8a187) : The SafeMath library, previously integral to the FarcanaToken and FarcanaVesting contracts was removed.

F-2023-0133 - Lack of Event Emissions in Key Functions of Vesting Contract - Info

Description:

The `addVestingSchedule` and `changeBeneficiaryAddress` functions in the vesting contract are crucial for managing vesting schedules and beneficiaries. However, both functions lack event emissions, which is a significant oversight. Events in smart contracts are essential for tracking changes on the blockchain, especially for key administrative actions.

Without events, tracking changes becomes challenging, reducing transparency and making it harder to verify actions retrospectively. This absence hinders external systems and interfaces from efficiently monitoring and reacting to important state changes in the contract.

Assets:

- FarcanaVesting.sol [https://github.com/farcana/smart-contracts]

Status: Fixed

Recommendations

Recommendation:

Introduce specific events for both functions to log significant activities:

- For `addVestingSchedule`, emit an event detailing the beneficiary address and key vesting parameters.
- For `changeBeneficiaryAddress`, emit an event capturing both the old and new beneficiary addresses.

Remediation (Revised commit: 3a8a187) : The FarcanaVesting contract now includes events in key functions

F-2023-0134 - State variables default visibility - Info

Description: Variable `tgetime` visibility is not specified. The default variable visibility in Solidity is `internal`. Specifying state variables visibility helps to catch incorrect assumptions about who can access the variable.

Assets:

- FarcanaVesting.sol [https://github.com/farcana/smart-contracts]

Status: Fixed

Recommendations

Recommendation: Specify variables as public, internal, or private. Explicitly define visibility for all state variables.

Remediation (Revised commit: 3a8a187) : The visibility is now explicitly declared as `internal`.

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.

Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
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.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details

Repository	https://github.com/farcana/smart-contracts
Commit	0c8f5193b9249d5b02fdb57de1ecdbf4ebe30dbb
Whitepaper	N/A
Requirements	NatSpec
Technical Requirements	N/A

Contracts in Scope

./contracts/FarcanaToken.sol

./contracts/FarcanaVesting.sol