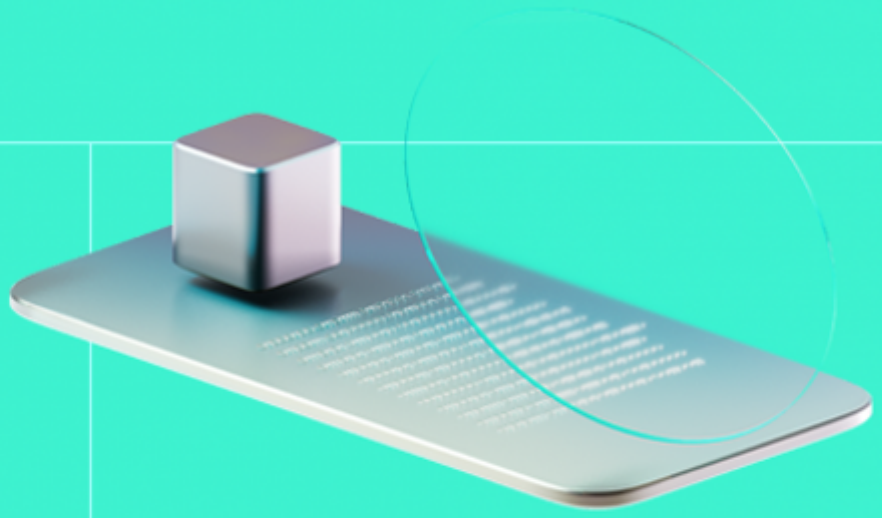
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer: MIN token

Date: 18/04/2024

We express our gratitude to the MIN Token team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

MIN Token project is an ERC20 based vesting project. Given a tokenomics structure with Token Generation Events (TGE), cliff and vesting periods.

Platform: Polygon

Language: Solidity

Tags: ERC20, Vesting

Timeline: 05/04/2024 -19/04/2024

Methodology: https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/baltarifcan/MINToken/?tab=readme-ov-file
Commit	0ec894f

Audit Summary

10/10	9/10	100%	10/10
Security Score	Code quality score	Test coverage	Documentation quality score

Total 9.8/10

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

6	5	0	1
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	1
High	1
Medium	1
Low	3

Vulnerability	Status
F-2024-2068 - Allows Adding a new Beneficiary to a Finished Vesting Schedule	Mitigated
F-2024-2057 - Missing Time Checks During Vesting Period Creation	Fixed
F-2024-2058 - Reserved Funds are not being Updated	Fixed
F-2024-2059 - Private Sale Allow Users to Deposit or Withdraw During a Vesting Period	Fixed
F-2024-2067 - Missing Cap Limit Check While Adding Beneficiaries to the Strategic Sale	Fixed
F-2024-2070 - Missing Free Amount Check When Users Are Added to Private Sale	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 MIN token
Audited By	Turgay Arda Usman
Approved By	Grzegorz Trawinski
Website	n/a
Changelog	09/04/2024 - Preliminary Report; 18/04/2024 - Final Report

Table of Contents

System Overview

MIN Token project is an ERC20 based vesting project. Given a tokenomics structure with Token Generation Events (TGE), cliff and vesting periods. It has the following contracts:

MINToken — simple ERC-20 token that mints all initial supply to a deployer. Additional minting is not allowed.

It has the following attributes:

- Name: MIN Token
- Symbol: MIN

MINVestingBase —This contract handles the base vesting schedule logic for the MIN token.

MINPrivateSwap —This contract is used for swapping tokens in a private sale.

MINStrategicSale — This contract manages the strategic sale of MIN tokens.

MINVesting — This contract handles the vesting schedule for the MIN token.

MINStructs — A library for managing MIN token related structures.

Privileged roles

- Set vesting schedules on MINVesting (Owners).
- Release vested amounts of MIN Token to wallets (Owners and buyers).
- Buy MIN Token with a set swappable token and rate (Buyers).
- Give up buying MIN Token through MINPrivateSwap and claim sent swappable token before sale ends (Buyers).
- Claim any unsold MIN Tokens and sale revenue in swappable token after sale ends (Owners).

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 **10** out of **10**.

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

Code quality

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

- The code mostly follows style guides and best practices
  - See low and informational issues for more information.
- The development environment is configured.

Test coverage

Code coverage of the project is **100%** (branch coverage),

- Deployment and basic user interactions are covered with tests.
- Negative cases coverage is not missed.

Security score

Upon auditing, the code was found to contain 1 critical, 1 high, 1 medium, and 3 low severity issues, leading to a security score of **0** out of 10. Upon the retest, all significant issues were fixed, 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 **9.8**. This score reflects the combined evaluation of documentation, code quality, test coverage, and security aspects of the project.

Risks

- The absence of restrictions on state variable modifications by the owner leads to arbitrary changes, affecting contract integrity and user trust, especially during critical operations like minting phases.
    - The implementation allows owner to withdraw MIN tokens and swap tokens any time without notifying anyone.

Findings

Vulnerability Details

F-2024-2059 - Private Sale Allow Users to Deposit or Withdraw During a Vesting Period - Critical

Description:

The `MINPrivateSwap.sol` contract, used for swapping tokens in a private sale. The contract allows for depositing and withdrawing tokens, with checks for sale end times. These operations are allowed according to the sale time but not the vesting duration or the cliff period. Meaning, if the sale period time collides with the vesting or cliff period a user will be able to deposit or withdraw funds to/from to the private vesting schedule and alter its balance.

```
function _updateBeneficiaryVestedAmount(address beneficiary, uint256
swapTokenBalance) private {
MINStructs.VestingSchedule memory vestingSchedule = MINStructs.Vesti
ngSchedule({
tgePermille: 0,
beneficiary: beneficiary,
startTimestamp: _privateSaleVestingSchedule.startTimestamp,
cliffDuration: _privateSaleVestingSchedule.cliffDuration,
vestingDuration: _privateSaleVestingSchedule.vestingDuration,
slicePeriodSeconds: _privateSaleVestingSchedule.slicePeriodSeconds,
totalAmount: 0,
releasedAmount: 0
});
if (swapTokenBalance > 0) {
vestingSchedule.totalAmount = ((swapTokenBalance * 100) / _ratioMinT
oSwap);
}
if (vestingSchedule.totalAmount > 0) {
_setVestingSchedule(vestingSchedule);
} else {
_removeVestingSchedule(beneficiary);
}
}
```

After each of these actions the released amount will be set to zero, as it can be seen above. In a case where, the sale time is accidentally set to a time between cliff period and the vesting deadline where some of the locked funds are already released by the beneficiary and a deposit or a withdrawal occurs, the released amount will be set to zero. Thus when the vesting schedule ends the user will be able to release more tokens than the schedule should.

In an other case a user can deposit to a finished private sale vesting schedule, set the released funds to zero and then trigger release function to get double funds.

Assets:

- ./vesting/MINPrivateSwap.sol [https://github.com/baltarifcan/MINToken/]

Status: Fixed

Classification

Severity: Critical

Impact: Likelihood [1-5]: 5

Impact [1-5]: 5

Exploitability [0-2]: 1

Complexity [0-2]: 1

Final Score: 3.5 (High)

Hacken Calculator Version: 0.6

Recommendations

Remediation: Make sure the sale period and vesting schedule periods are correctly aligned or the re-implement the `_updateBeneficiaryVestedAmount()` function so that it tracks the released amount

Remediation (revised commit: 7823a9a): The following check is implemented to align sale and vesting periods.

```
require(
block.timestamp + saleDuration <= privateSaleVestingSchedule.startTi
mestamp,
"MINPrivateSwap: sale must end before cliff and vesting starts"
);
```

Evidences

PoC

Reproduce:

- Initial setup
  - **Sale Duration:** 1 week (7 days) from cliff period
  - **Vesting Schedule:**
    - Cliff Period: 3 months from today
    - Vesting Duration: 1 year from today
    - total amount: 1,000,000 MIN tokens
- On day 1 of the sale, Alice releases 300,000 MIN tokens.
  - total amount: 1,000,000
  - released amount: 300,000
- On day 3 of the sale, Alice deposits 10,000 swap tokens. The contract calculates that Alice is eligible to receive 1,000,000 MIN tokens (10,000 tokens * 100 MIN/tokens).

    - due to the `_updateBeneficiaryVestedAmount()` function the released amount is set to zero.
        - total amount: 2,000,000
        - released amount: 0
- When the vesting schedule ends and Alice releases her funds, Alice will be able to get all the 2,000,000 MIN tokens + 300,000 she released before.

F-2024-2058 - Reserved Funds are not being Updated - High

Description:

The `_totalReservedAmount` variable is being used to track the total amount reserved for the beneficiaries. This variable is included into various key calculations such as withdrawable token amount calculation and global limit checks in during new beneficiary addition, and global limit checks while creating a new vesting schedule. However the only way to update this variable is through the following function, `addToTotalReservedAmount()`:

```
function addToTotalReservedAmount(uint256 amount) internal {
_totalReservedAmount += amount;
emit TotalReservedAmountUpdated(_totalReservedAmount);
}
```

As it can be seen, this function is internal, meaning the owner cannot call it manually, and it is only being called in the strategic vesting schedules. That means for all the other vesting schedules the reserved amount will not be applied to the calculations and the system will allow to more withdrawals, beneficiary, or vesting schedules than it should.

This will lead to imbalances.

Assets:

- ./vesting/MINVesting.sol [https://github.com/baltarifcan/MINToken/]

Status: Fixed

Classification

Severity: High

Impact:

Likelihood [1-5]: 4

Impact [1-5]: 5

Exploitability [0-2]: 0

Complexity [0-2]: 0

Final Score: 4.5 (High)

Hacken Calculator Version: 0.6

Recommendations

Remediation: Update the reserved amount for every new beneficiary or vesting schedule addition.

Remediation (revised commit: 7823a9a): The total reserved amount is being updated via `addToTotalReservedAmount()` function in each vesting creation.

Evidences

PoC

Reproduce:

For the following vesting schedules:

```
strategic: {
tgePermille: 100,
beneficiary: WALLETS.strategic,
startTimestamp: START_DATE,
cliffDuration: 2 * MONTH,
vestingDuration: 18 * MONTH,
slicePeriodSeconds: MONTH,
totalAmount: BigInt(7_500_000) * 10n ** 18n,
releasedAmount: 0,
},
private: {
tgePermille: 0,
beneficiary: WALLETS.private,
startTimestamp: START_DATE,
cliffDuration: 4 * MONTH,
vestingDuration: 12 * MONTH,
slicePeriodSeconds: MONTH,
totalAmount: BigInt(1_500_000) * 10n ** 18n,
releasedAmount: 0,
},
public: {
tgePermille: 100,
beneficiary: WALLETS.public,
startTimestamp: START_DATE,
cliffDuration: 2 * MONTH,
vestingDuration: 18 * MONTH,
slicePeriodSeconds: MONTH,
totalAmount: BigInt(27_000_000) * 10n ** 18n,
releasedAmount: 0,
},
enGaranti: {
tgePermille: 0,
beneficiary: WALLETS.enGaranti,
startTimestamp: START_DATE,
cliffDuration: 3 * MONTH,
vestingDuration: 60 * MONTH,
slicePeriodSeconds: MONTH,
totalAmount: BigInt(30_000_000) * 10n ** 18n,
releasedAmount: 0,
},
operations: {
tgePermille: 0,
beneficiary: WALLETS.operations,
startTimestamp: START_DATE,
cliffDuration: 6 * MONTH,
vestingDuration: 48 * MONTH,
slicePeriodSeconds: MONTH,
totalAmount: BigInt(24_000_000) * 10n ** 18n,
releasedAmount: 0,
},
marketingAndRewards: {
tgePermille: 15,
beneficiary: WALLETS.marketingAndRewards,
startTimestamp: START_DATE,
cliffDuration: 0 * MONTH,
vestingDuration: 59 * MONTH,
slicePeriodSeconds: MONTH,
totalAmount: BigInt(60_000_000) * 10n ** 18n,
releasedAmount: 0,
},
devTeam: {
```

```
tgePermille: 0,
beneficiary: WALLETS.devTeam,
startTimestamp: START_DATE,
cliffDuration: 3 * MONTH,
vestingDuration: 60 * MONTH,
slicePeriodSeconds: MONTH,
totalAmount: BigInt(30_000_000) * 10n ** 18n,
releasedAmount: 0,
},
reserve: {
tgePermille: 0,
beneficiary: WALLETS.reserve,
startTimestamp: START_DATE,
cliffDuration: 48 * MONTH,
vestingDuration: 12 * MONTH,
slicePeriodSeconds: MONTH,
t
```

See more

F-2024-2070 - Missing Free Amount Check When Users Are Added to Private Sale - Medium

Description:

The private sale schedule allows its users to deposit or withdraw funds to their schedule through the `deposit()` and `withdraw()` functions. These functions call the `_updateBeneficiaryVestedAmount()` function which creates or updates the record for the private sale.

```
function deposit(uint256 amount) public onlyBeforeSaleEnd {
require(
(((_swapToken.balanceOf(address(this)) + amount) * 100) / _ratioMinT
oSwap) <= _maxMinToken,
"MINPrivateSwap: not enough MIN tokens to buy for the swap tokens"
);
....
}

function _updateBeneficiaryVestedAmount(address beneficiary, uint256
swapTokenBalance) private {
MINStructs.VestingSchedule memory vestingSchedule = MINStructs.Vesti
ngSchedule({
tgePermille: 0,
beneficiary: beneficiary,
startTimestamp: _privateSaleVestingSchedule.startTimestamp,
cliffDuration: _privateSaleVestingSchedule.cliffDuration,
vestingDuration: _privateSaleVestingSchedule.vestingDuration,
slicePeriodSeconds: _privateSaleVestingSchedule.slicePeriodSeconds,
totalAmount: 0,
releasedAmount: 0
});
if (swapTokenBalance > 0) {
vestingSchedule.totalAmount = ((swapTokenBalance * 100) / _ratioMinT
oSwap);
}
if (vestingSchedule.totalAmount > 0) {
_setVestingSchedule(vestingSchedule);
} else {
_removeVestingSchedule(beneficiary);
}
}
```

As it can be seen this function just updates the related mapping and does not check if the caller is actually a private sale member or not. This means that any user can deposit funds to the private sale and turn their schedule into a private sale schedule. This will cause imbalances in the tokenomics design.

The tokenomics design allows 1.500.000 MIN tokens for the private sale schedule, However the implementation does not check if there are enough free amount to create a new vesting schedule with the given settings. It only checks if the deposited amount does not exceed the total amount.

In addition to that since the implementation does not check the deadlines, a finished vesting schedule user, can deposit funds and become a private sale member then wait for its release and get more funds.

Assets:

- ./vesting/MINPrivateSwap.sol [https://github.com/baltarifcan/MINToken/]

Status: Fixed

Classification

Severity: Medium

Impact: Likelihood [1-5]: 4

Impact [1-5]: 5

Exploitability [0-2]: 1

Complexity [0-2]: 1

Final Score: 3.2 (Medium)

Hacken Calculator Version: 0.6

Recommendations

Remediation: Allow only related parties to access private sale functions.

F-2024-2057 - Missing Time Checks During Vesting Period Creation - Low

Description:

The `_setVestingSchedule()` function allows owner to create vesting schedules via functions. However, the system does not check if these new vesting schedules are being created in a past date or their durations are long enough to align with the tokenomics provided.

```
function _setVestingSchedule(MINStructs.VestingSchedule memory vesti
ngSchedule) internal {
require(vestingSchedule.beneficiary != address(0), "MINVesting: bene
ficiary address cannot be zero");
require(vestingSchedule.totalAmount > 0, "MINVesting: total amount m
ust be greater than zero");
require(vestingSchedule.slicePeriodSeconds > 0, "MINVesting: slice p
eriod must be greater than zero");
require(
vestingSchedule.vestingDuration > 0 &&
vestingSchedule.slicePeriodSeconds <= vestingSchedule.vestingDuratio
n,
"MINVesting: vesting duration must be greater than zero and slice pe
riod"
);


_vestingSchedules[vestingSchedule.beneficiary] = vestingSchedule;
emit VestingScheduleSet(vestingSchedule.beneficiary, vestingSchedule
);
}
```

Assets:

- ./utils/MINVestingBase.sol [https://github.com/baltarifcan/MINToken/]

Status: Fixed

Classification

Severity: Low

Impact:

Likelihood [1-5]: 2

Impact [1-5]: 3

Exploitability [0-2]: 0

Complexity [0-2]: 1

Final Score: 2.3 (Low)

Hacken Calculator Version: 0.6

Recommendations

Remediation: Implement deadline checks.

Remediation (revised commit: 7823a9a): Deadline check has been implemented

F-2024-2067 - Missing Cap Limit Check While Adding Beneficiaries to the Strategic Sale - Low

Description:

The `addBeneficiary()` function, ads a beneficiary to the strategic sale. It does not check if the given amount is greater than the MIN token supply or the pre determined vesting schedule limit.

```
function addBeneficiary(address beneficiary, uint256 amount) public
onlyOwner {
require(
getVestingSchedule(beneficiary).beneficiary == address(0),
"MINStrategicSale: beneficiary already exists"
);

require(amount > 0, "MINStrategicSale: amount must be greater than 0
");
require(
amount <= getToken().balanceOf(address(this)) - getTotalReservedAmou
nt(),
"MINStrategicSale: amount must be less than or equal to contract bal
ance"
);

MINStructs.VestingSchedule memory vestingSchedule = MINStructs.Vesti
ngSchedule({
tgePermille: _strategicSaleVestingSchedule.tgePermille,
beneficiary: beneficiary,
startTimestamp: _strategicSaleVestingSchedule.startTimestamp,
cliffDuration: _strategicSaleVestingSchedule.cliffDuration,
vestingDuration: _strategicSaleVestingSchedule.vestingDuration,
slicePeriodSeconds: _strategicSaleVestingSchedule.slicePeriodSeconds
,
totalAmount: amount,
releasedAmount: 0
});

_setVestingSchedule(vestingSchedule);
addToTotalReservedAmount(amount);

emit BeneficiaryAdded(beneficiary, amount);
}
```

Assets:

- ./vesting/MINStrategicSale.sol [https://github.com/baltarifcan/MINToken/]

Status: Fixed

Classification

Severity: Low

Impact:

Likelihood [1-5]: 3

Impact [1-5]: 3

Exploitability [0-2]: 2

Complexity [0-2]: 0

Final Score: 2.1 (Low)

Recommendations

Remediation: Implement cap limitations.

Remediation (revised commit: 7823a9a): The following check is implemented so now, the beneficiaries are added if there are enough free funds.

```
require( amount <= computeWithdrawableMintokens(),
"MINStrategicSale: amount must be less than or equal to
contract balance" );
```

F-2024-2068 - Allows Adding a new Beneficiary to a Finished Vesting Schedule - Low

Description:

The `addBeneficiary()` function, ads a beneficiary to the strategic sale. However, it never checks if the vesting period has ended or not. This allows users to add beneficiaries to expired vest.ng schedules.

```
function addBeneficiary(address beneficiary, uint256 amount) public
onlyOwner {
require(
getVestingSchedule(beneficiary).beneficiary == address(0),
"MINStrategicSale: beneficiary already exists"
);

require(amount > 0, "MINStrategicSale: amount must be greater than 0
");
require(
amount <= getToken().balanceOf(address(this)) - getTotalReservedAmou
nt(),
"MINStrategicSale: amount must be less than or equal to contract bal
ance"
);

MINStructs.VestingSchedule memory vestingSchedule = MINStructs.Vesti
ngSchedule({
tgePermille: _strategicSaleVestingSchedule.tgePermille,
beneficiary: beneficiary,
startTimestamp: _strategicSaleVestingSchedule.startTimestamp,
cliffDuration: _strategicSaleVestingSchedule.cliffDuration,
vestingDuration: _strategicSaleVestingSchedule.vestingDuration,
slicePeriodSeconds: _strategicSaleVestingSchedule.slicePeriodSeconds
,
totalAmount: amount,
releasedAmount: 0
});

_setVestingSchedule(vestingSchedule);
addToTotalReservedAmount(amount);

emit BeneficiaryAdded(beneficiary, amount);
}
```

Assets:

- ./vesting/MINStrategicSale.sol [https://github.com/baltarifcan/MINToken/]

Status: Mitigated

Classification

Severity: Low

Impact:

Likelihood [1-5]: 3

Impact [1-5]: 3

Exploitability [0-2]: 2

Complexity [0-2]: 0

Final Score: 2.1 (Low)

Recommendations

Remediation: Implement deadline checks.

Remediation (Mitigated): This is an intended behaviour according to the client's business logic.

Observation Details

F-2024-2053 - Missing Zero Address Violation - Info

Description:

In Solidity, the Ethereum address `0x0000000000000000000000000000000000000000` is known as the "**zero address"**. This address has significance because it is the default value for uninitialized address variables and is often used to represent an invalid or non-existent address.

The "**Missing zero address control**" issue arises when a Solidity smart contract does not properly check or prevent interactions with the zero address, leading to unintended behavior.

For instance, consider a contract that includes a function to change its owner. This function is crucial, as it determines who has administrative access. However, if this function lacks proper validation checks, it might inadvertently permit the setting of the owner to the zero address. Consequently, the administrative functions will become unusable.

Assets:

- ./vesting/MINStrategicSale.sol [https://github.com/baltarifcan/MINToken/]
- ./vesting/MINPrivateSwap.sol [https://github.com/baltarifcan/MINToken/]
- ./utils/MINVestingBase.sol [https://github.com/baltarifcan/MINToken/]

Status: Fixed

Recommendations

Remediation:

Implement zero address validation for the given parameters. This can be achieved by adding require statements that ensure address parameters are not the zero address.

Remediation (revised commit: 7823a9a): Zero address checks have been implemented.

F-2024-2054 - Redundant Function - Info

Description: The `getCurrentTime()` function is redundant, as they only return the global variables `block.timestamp`.

Assets:

- ./utils/MINVestingBase.sol [https://github.com/baltarifcan/MINToken/]

Status: Fixed

Recommendations

Remediation: Remove the redundant function.

Remediation (revised commit: 7823a9a): The redundant function is removed

F-2024-2055 - Memory Exhaustion Risk Due to Deletion Logic - Info

Description:

The `_removeVestingSchedule()` function aims to remove the vesting schedule for a beneficiary. To do that it benefits from the delete keyword. This keyword does not actually deletes the records in a struct, it sets them to their default value. Since the record in examination is a struct all of its values will be set to default but the record will stay there. This can cause unnecessary memory load on the system.

```
function _removeVestingSchedule(address beneficiary) internal {
delete _vestingSchedules[beneficiary];
}
```

Assets:

- ./utils/MINVestingBase.sol [https://github.com/baltarifcan/MINToken/]

Status: Accepted

Recommendations

Remediation:

Adopt tombstone pattern or prefer sparse mapping libraries such asOpenZeppelin Upgrades.

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 **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

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/baltarifcan/MINToken/?tab=readme-ov-file
Commit	0ec894f
Whitepaper	n/a
Requirements	https://github.com/baltarifcan/MINToken/?tab=readme-ov-file
Technical Requirements	https://github.com/baltarifcan/MINToken/?tab=readme-ov-file

Contracts in Scope

contracts/token/MINToken.sol

contracts/vesting/MINVesting.sol

contracts/vesting/MINStrategicSale.sol

contracts/vesting/MINPrivateSwap.sol

contracts/utils/MINVestingBase.sol

contracts/utils/MINStructs.sol