CERTIK

Security Assessment

Monsta Infinite

Sept 7th, 2021

Table of Contents

Summary

This report has been prepared for Monsta Infinite to discover issues and vulnerabilities in the source code of the Monsta Infinite project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	Monsta Infinite
Platform	BSC
Language	Solidity
Codebase	https://gitlab.com/monsta-infinite/moni-smart-contracts/-/tree/moni-presale
Commit	• c3f7dcff0bfaff835762114afed919d78a8acbef • f592f69ae48d2390cc2f9fca4b60faefcb9682c8

Audit Summary

Delivery Date	Sept 07, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
● Critical	0	0	0	0	0	0
● Major	3	0	0	0	3	0
● Medium	4	0	0	4	0	0
● Minor	3	0	0	3	0	0
● Informational	11	0	0	11	0	0
● Discussion	0	0	0	0	0	0

Audit Scope

ID			File	SHA256 Checksum

Understandings

Overview

Monsta Finance has created a decentralized game universe where anyone can earn tokens through playing the game competitively.

The total supply of `MONI` token is 270,000,000. The tokenomic is as below:

- Private Sale: 6% (16,200,000)
- Pre Sale 1: 0.2% (540,000)
- Pre Sale 2: 2.8% (7,560,000)
- IDO: 1% (2,700,000)
- Advisor: 5% (13,500,000)
- Team: 16% (43,200,000)
- Marketing: 1% (2,700,000)
- Ecosystem Found: 5% (13,500,00)
- B.Launchpad: 5% (13,500,000)
- Gamplay: 25% (67,500,00)

The project applies Gnosis multisig wallet to require 2 (out of 3) signatures to confirm every transaction in order to execute it, which helps prevent unauthorized access to executing certain functions.

- **0x9573c88aE3e37508f87649f87c4dd5373C9F31e0** is the multisig wallet for the BEP20 token contract
- **0xaa552Ccd7C784470733e924d053bf7B69f25Cba4** is the multisig wallet for the Presale1 contract
- **0xa1554142321bdBf9513aB826bf6685F5c2b787DA** is the multisig wallet for the Presale contract
- **0x46FfD72dAD2119e112d98933B93b570bA0d7E71B** is the multisig wallet for the Owner
- **0x467db17EbC0FB29510a63B31332446C92DFF44fE** is the multisig wallet for the Private Sale contract
- **0xDC6FC5e0111dBdC6111AdF2ca11B7C4F234d49C6** is the multisig wallet for the IDO
- **0x15B3A05aC648881DeB1fD1156eF5f0C65f46aceB** is the multisig wallet for the Advisors
- **0xa2867165F5b864cd5ac08938f0afE1C69e5Da204** is the multisig wallet for the Team
- **0x0210a14e724c7a6769c393853be6cC0f1Dff2687** is the multisig wallet for the Marketing
- **0x9CC234DE2CF4b0C9a1C64Bb3E4f96d6aa1176698** is the multisig wallet for the Liquidity
- **0x72Dc63161173B7371fbDdaF3a67A6A990b5c9938** is the multisig wallet for the Ecosystem Fund
- **0xA71B91f139Fc59C22b4c7DC91CDFdaadAEB10E0C** is the multisig wallet for the B Launchpad
- **0x93f2440bE026F37B7cd3Fc50281253a54Daaa7B9** is the multisig wallet for the Gameplay
- **0xA8541E4d7b548B8ab05db477f1C14442D4931A99** is the multisig wallet for the Staking

Dependencies

There are a few depending injection contracts or addresses in the current project:

- `advisorAddress`, `teamAddress`, `marketingAddress`, `ecosystemFundAddress`, `gameplayAddress`, `stakingAddress`, `0xe0E7a8b6Fcd7c37CE6768D57DCf18b8100e5C8D4`, `0x1382E0Ac5e01D4181483bEB384c0e09dd671717E`, `0xDA18Eb309D418Ca9Fe2929a233fd67DC1f3EF489`, `0xe2665d26A2B136c916f4BaCF4b61D5c529D37f5b`, `preSale1Contract`, and `preSale2Contract` for the contract `MoniToken`;
- `_redemptionAddress` for the contract `MonstaPresale`;
- `_erc20Contract` for the contract `MoniPreSale1`;
- `_erc20Contract` for the contract `MoniPreSale2`.

We assume these contracts or addresses are valid and non-vulnerable actors and implementing proper logic to collaborate with the current project.

Privileged Functions

In the contract `MoniToken`, the role `admin` has the authority over the following function:

- `MoniToken.transferOwnership()` to transfer the `admin` role;
- `MoniToken.setPreSale1ContractNotYetSet()` to set the first pre-sale contract;
- `MoniToken.setPreSale2ContractNotYetSet()` to set the second pre-sale contract;
- `MoniToken.pause()` to pause the contract;
- `MoniToken.unpause()` to unpause the contract;
- `MoniToken.pausedAddress()` to pause certain address;
- `MoniToken.unPausedAddress()` to unpause certain address.

In the contract `MoniPreSale1`, the role `admin` has the authority over the following function:

- `MoniPreSale1.transferOwnership()` to transfer the `admin` role;
- `MoniPreSale1.addDepositAddress()` to add deposit addresses;
- `MoniPreSale1.removeAllDepositAddress()` to remove depossit addresses;
- `MoniPreSale1.withdrawAll()` to withdraw all deposited BNB.

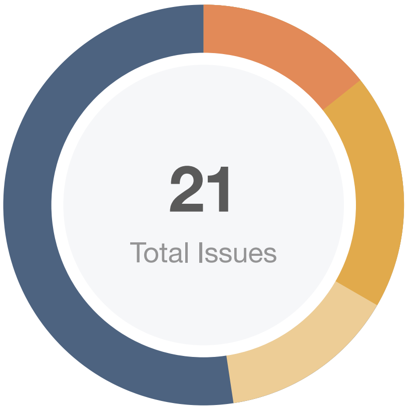
In the contract `MoniPreSale2`, the role `admin` has the authority over the following function:

- `MoniPreSale2.transferOwnership()` to transfer the `admin` role;
- `MoniPreSale2.addDepositAddress()` to add deposit addresses;
- `MoniPreSale2.removeAllDepositAddress()` to remove depossit addresses;
- `MoniPreSale2.endPreSale2Earlier()` to end the presale earlier than the schedule;

- `MoniPreSale2.withdrawAll()` to withdraw all deposited BNB.

To improve the trustworthiness of the project, any dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should be also considered to move to the execution queue of the `Timelock` contract.

Findings



🟥	**Critical**	**0** (0.00%)
🟧	**Major**	**3** (14.29%)
🟨	**Medium**	**4** (19.05%)
🟨	**Minor**	**3** (14.29%)
🟦	**Informational**	**11** (52.38%)
🟩	**Discussion**	**0** (0.00%)

ID	Title	Category	Severity	Status
MPC-01	Centralization Risk	**Centralization / Privilege**	● **Major**	Partially Resolved
MPC-02	Potential Resource Exhaustion	Volatile Code	● Medium	ⓘ Acknowledged
MPC-03	Reentrancy Attack Risk	Logical Issue	● Medium	ⓘ Acknowledged
MPC-04	`SafeMath` Not Used	Mathematical Operations	● Minor	ⓘ Acknowledged
MPC-05	Missing Error Message	Coding Style	● Informational	ⓘ Acknowledged
MPC-06	Missing Event Emissions for Significant Transactions	Coding Style	● Informational	ⓘ Acknowledged
MPC-07	Redundant Statement	Coding Style	● Informational	ⓘ Acknowledged
MPC-08	Return Value Not Handled	Volatile Code	● Informational	ⓘ Acknowledged
MPC-09	Deposit Amount Can Be Less Than `_minimumDepositBNBAmount`	Logical Issue	● Informational	ⓘ Acknowledged
MPS-01	Centralization Risk	**Centralization / Privilege**	● **Major**	Partially Resolved
MPS-02	Potential Reentrancy Attack	Logical Issue	● Medium	ⓘ Acknowledged
MPS-03	Potential Resource Exhaustion	Volatile Code	● Medium	ⓘ Acknowledged
MPS-04	`SafeMath` Not Used	Mathematical Operations	● Minor	ⓘ Acknowledged

ID	Title	Category	Severity	Status
MPS-05	Missing Event Emissions for Significant Transactions	Coding Style	● Informational	ⓘ Acknowledged
MPS-06	Redundant Statement	Coding Style	● Informational	ⓘ Acknowledged
MPS-07	Return Value Not Handled	Volatile Code	● Informational	ⓘ Acknowledged
MPS-08	Missing Error Messages	Coding Style	● Informational	ⓘ Acknowledged
MTC-01	Centralization Risk	**Centralization / Privilege**	● **Major**	◔ Partially Resolved
MTC-02	Missing Error Messages	Coding Style	● Minor	ⓘ Acknowledged
MTC-03	Variable Could Be Declared as `constant`	Gas Optimization, Coding Style	● Informational	ⓘ Acknowledged
MTC-04	Missing Event Emissions for Significant Transactions	Coding Style	● Informational	ⓘ Acknowledged

MPC-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	presale/MoniPresale1.sol: 51, 57, 71, 142	Partially Resolved

Description

In the contract `MoniPreSale1`, the role `admin` has the authority over the following function:

- `MoniPreSale1.transferOwnership()` to transfer the `admin` role;
- `MoniPreSale1.addDepositAddress()` to add deposit addresses;
- `MoniPreSale1.removeAllDepositAddress()` to remove depossit addresses;
- `MoniPreSale1.withdrawAll()` to withdraw all deposited BNB.

Any compromise to the `admin` account may allow the hacker to take advantage of this and manipulate the protocol.

Recommendation

We advise the client to carefully manage the `admin` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Indicatively, here is some feasible suggestions that would also mitigate the potential risk at the different level in term of short-term and long-term:

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

The Monsta Team applies a Gnosis multi-sig wallet 0xaa552Ccd7C784470733e924d053bf7B69f25Cba4 for the contract admin. It requires 2 (out of 3) team members to confirm every transaction in order to execute it, which helps prevent unauthorized access to executing the aforementioned functions.

In addition, we recommend any plan to invoke the aforementioned function should be also considered to move to the execution queue of the `Timelock` contract and dynamic runtime updates in the project should be notified to the community ahead.

MPC-02 | Potential Resource Exhaustion

Category	Severity	Location	Status
Volatile Code	● Medium	presale/MoniPresale1.sol: 71~83	ⓘ Acknowledged

Description

The function `removeAllDepositAddress(uint)` is designed to remove and unwhitelist certain addresses from deposit.

```
71    function removeAllDepositAddress(uint number) external onlyAdmin {
72      require(block.timestamp < _SEP_15_2021_00_00_00, "Presale2 already started");
73      uint i = _startDepositAddressIndex;
74      uint last = i + number;
75      if (last > _depositAddressesNumber) last = _depositAddressesNumber;
76      for (; i < last; i++) {
77        _depositAddressesStatus[_depositAddresses[i]] = false;
78        _depositAddresses[i] = address(0);
79      }
80      _startDepositAddressIndex = i;
81      _distributeFirstIndex = i;
82      _distributeSecondIndex = i;
83      _distributeThirdIndex = i;
84    }
```

However, the code block above only sets the "deleted" addresses to be `address(0)` (L78), without deleting the memory space. Therefore, it might be exposing a Denial-of-Service attack vector that can be exploited. For example, if an attacker can cheat the `admin` of the contract to keep adding and removing his addresses (i.e., executing `addDepositAddress()` and `removeAllDepositAddress()` repeatedly), the array `_depositAddresses` might be filled with `address(0)`. In an extreme case, if the attacker could fill all the `_depositAddresses` arrays with `address(0)`, it could cause some unexpected loss to the project.

On the other hand, when distributing tokens after presale, the `_distribute()` function would iterate recursively over `_depositAddresses`, which might contain multiple `address(0)`. This would cause extra gas costs to the project.

Recommendation

We advise the client to delete the addresses by using `delete _depositAddresses[i]` and decrease the index `_depositAddressesNumber` by 1.

MPC-03 | Reentrancy Attack Risk

Category	Severity	Location	Status
Logical Issue	● Medium	presale/MoniPresale1.sol: 112, 118	ⓘ Acknowledged

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

MPC-04 | `SafeMath` Not Used

Category	Severity	Location	Status
Mathematical Operations	● Minor	presale/MoniPresale1.sol: 63, 74, 128	ⓘ Acknowledged

Description

The following expressions do not check arithmetic overflow. Such unsafe math operations may cause unexpected behavior if unusual parameters are given.

```
63        depositAddressesNumber++;
```

```
74        uint last = i + number;
```

```
128        uint last = i + number;
```

Recommendation

We advise the client to use OpenZeppelin's SafeMath library for the aforementioned mathematical operations.

Reference: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/math/SafeMath.sol

MPC-05 | Missing Error Message

Category	Severity	Location	Status
Coding Style	● Informational	presale/MoniPresale1.sol: 43	ⓘ Acknowledged

Description

The **require** can be used to check for conditions and throw an exception if the condition is not met. It is better to provide a string message containing details about the error that will be passed back to the caller.

Recommendation

We recommend adding appropriate error messages in the aforementioned **require** statements.

MPC-06 | Missing Event Emissions for Significant Transactions

Category	Severity	Location	Status
Coding Style	● Informational	presale/MoniPresale1.sol: 51, 57, 71, 142	ⓘ Acknowledged

Description

The functions that affect the status of sensitive variables should be able to emit events as notifications to users. For example,

- `MoniPreSale1.transferOwnership()` to transfer the `admin` role;
- `MoniPreSale1.addDepositAddress()` to add deposit addresses;
- `MoniPreSale1.removeAllDepositAddress()` to remove depossit addresses;
- `MoniPreSale1.withdrawAll()` to withdraw all deposited BNB.

Recommendation

We recommend emitting events for all the essential state variables that are possible to be changed during the runtime.

MPC-07 | Redundant Statement

Category	Severity	Location	Status
Coding Style	● Informational	presale/MoniPresale1.sol: 58~66	ⓘ Acknowledged

Description

The variable `depositAddressesNumber` declared in L58 is redundant since it can be replaced by the global variable `_depositAddressesNumber`.

Recommendation

We advise revising the code and removing the redundant part. For example,

```
57 function addDepositAddress(address[] calldata depositAddresses) external onlyAdmin {
58   for (uint i = 0; i < depositAddresses.length; i++) {
59     if (!_depositAddressesStatus[depositAddresses[i]]) {
60       _depositAddresses[_depositAddressesNumber] = depositAddresses[i];
61       _depositAddressesStatus[depositAddresses[i]] = true;
62       _depositAddressesNumber++;
63     }
64   }
65 }
```

MPC-08 | Return Value Not Handled

Category	Severity	Location	Status
Volatile Code	● Informational	presale/MoniPresale1.sol: 136	ⓘ Acknowledged

Description

The return value of `transferPresale1()` is not properly handled.

```
136         erc20Contract.transferPresale1(depositor, deposited.mul(amount));
```

`transferPresale1()` is not void-return functions per `IErc20Contract` interface. Ignoring the return value of the function might cause some unexpected exceptions, especially if the called function does not revert automatically on failure.

Recommendation

We recommend checking the return value of the aforementioned function and handling both success and failure cases based on the business logic.

MPC-09 | Deposit Amount Can Be Less Than `_minimumDepositBNBAmount`

Category	Severity	Location	Status
Logical Issue	● Informational	presale/MoniPresale1.sol: 96	ⓘ Acknowledged

Description

In the `MoniPresale1.receive()` function, the `msg.value` is required to be greater than the `_minimumDepositBNBAmount`. However, the actual deposited amount be can be less than that.

For example, a user sends 2 `BNB` to this contract and the `_totalAddressesDepositAmount` is currently 199.5 `BNB`. In this case, the contract would send back the extra amount of `BNB`, i.e., 1.5 `BNB`, to the user, resulting in depositing the amount (0.5 `BNB`) less than the `_minimumDepositBNBAmount` (1 `BNB`).

Recommendation

We advise the team to review the design and ensure it is the intended design.

Alleviation

(**Monsta Team Response**)

It is the intended design.

MPS-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	presale/MoniPreSale2.sol: 51, 57, 71, 162, 167	Partially Resolved

Description

In the contract `MoniPreSale2`, the role `admin` has the authority over the following function:

- `MoniPreSale2.transferOwnership()` to transfer the `admin` role;
- `MoniPreSale2.addDepositAddress()` to add deposit addresses;
- `MoniPreSale2.removeAllDepositAddress()` to remove depossit addresses;
- `MoniPreSale2.endPreSale2Earlier()` to end the presale earlier than the schedule;
- `MoniPreSale2.withdrawAll()` to withdraw all deposited BNB.

Any compromise to the `admin` account may allow the hacker to take advantage of this and manipulate the protocol.

Recommendation

We advise the client to carefully manage the `admin` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Indicatively, here is some feasible suggestions that would also mitigate the potential risk at the different level in term of short-term and long-term:

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

The Monsta Team is planning to apply a Gnosis multi-sig wallet (address is unknown) for the contract admin. It requires 2 (out of 3) team members to confirm every transaction in order to execute it, which helps prevent unauthorized access to executing the aforementioned functions.

In addition, we recommend any plan to invoke the aforementioned function should be also considered to move to the execution queue of the `Timelock` contract and dynamic runtime updates in the project should be notified to the community ahead.

MPS-02 | Potential Reentrancy Attack

Category	Severity	Location	Status
Logical Issue	● Medium	presale/MoniPreSale2.sol: 102, 122, 128, 134	ⓘ Acknowledged

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

MPS-03 | Potential Resource Exhaustion

Category	Severity	Location	Status
Volatile Code	● Medium	presale/MoniPreSale2.sol: 71~84	ⓘ Acknowledged

Description

The function `removeAllDepositAddress(uint)` is designed to remove and unwhitelist certain addresses from deposit.

```
71    function removeAllDepositAddress(uint number) external onlyAdmin {
72      require(block.timestamp < _SEP_15_2021_00_00_00, "Presale2 already started");
73      uint i = _startDepositAddressIndex;
74      uint last = i + number;
75      if (last > _depositAddressesNumber) last = _depositAddressesNumber;
76      for (; i < last; i++) {
77        _depositAddressesStatus[_depositAddresses[i]] = false;
78        _depositAddresses[i] = address(0);
79      }
80      _startDepositAddressIndex = i;
81      _distributeFirstIndex = i;
82      _distributeSecondIndex = i;
83      _distributeThirdIndex = i;
84    }
```

However, the code block above only sets the "deleted" addresses to be `address(0)` (L78), without deleting the memory space. Therefore, it might be exposing a Denial-of-Service attack vector that can be exploited. For example, if an attacker can cheat the `admin` of the contract to keep adding and removing his addresses (i.e., executing `addDepositAddress()` and `removeAllDepositAddress()` repeatedly), the array `_depositAddresses` might be filled with `address(0)`. In an extreme case, if the attacker could fill all the `_depositAddresses` arrays with `address(0)`, it could cause some unexpected loss to the project.

On the other hand, the `_distribute()` function which distributes the tokens after presale would iterate recursively over `_depositAddresses`, which might contain multiple `address(0)`. This would cause extra gas costs.

Recommendation

We advise the client to delete the addresses by using `delete _depositAddresses[i]` and decrease the index `_depositAddressesNumber` by 1.

MPS-04 | `SafeMath` Not Used

Category	Severity	Location	Status
Mathematical Operations	● Minor	presale/MoniPreSale2.sol: 63, 74, 108, 144	ⓘ Acknowledged

Description

The following expressions do not check arithmetic overflow. Such unsafe math operations may cause unexpected behavior if unusual parameters are given.

```
63      depositAddressesNumber++;
```

```
74      uint last = i + number;
```

```
108      uint last = i + number;
```

```
144      uint last = i + number;
```

Recommendation

We advise the client to use OpenZeppelin's SafeMath library for the aforementioned mathematical operations.

Reference: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/math/SafeMath.sol

MPS-05 | Missing Event Emissions for Significant Transactions

Category	Severity	Location	Status
Coding Style	● Informational	presale/MoniPreSale2.sol: 51, 57, 71, 162, 167	ⓘ Acknowledged

Description

The function that affects the status of sensitive variables should be able to emit events as notifications to users. For example,

- `MoniPreSale2.transferOwnership()` to transfer the `admin` role;
- `MoniPreSale2.addDepositAddress()` to add deposit addresses;
- `MoniPreSale2.removeAllDepositAddress()` to remove depossit addresses;
- `MoniPreSale2.endPreSale2Earlier()` to end the presale earlier than the schedule.
- `MoniPreSale2.withdrawAll()` to withdraw all deposited BNB.

Recommendation

We recommend emitting events for all the essential state variables that are possible to be changed during the runtime.

MPS-06 | Redundant Statement

Category	Severity	Location	Status
Coding Style	● Informational	presale/MoniPreSale2.sol: 58	ⓘ Acknowledged

Description

The variable `depositAddressesNumber` declared in L58 is redundant since it can be replaced by the global variable `_depositAddressesNumber`.

Recommendation

We advise revising the code and removing the redundant part. For example,

```
 57   function addDepositAddress(address[] calldata depositAddresses) external onlyAdmin
{
 58     for (uint i = 0; i < depositAddresses.length; i++) {
 59       if (!_depositAddressesStatus[depositAddresses[i]]) {
 60         _depositAddresses[_depositAddressesNumber] = depositAddresses[i];
 61         _depositAddressesStatus[depositAddresses[i]] = true;
 62         _depositAddressesNumber++;
 63       }
 64     }
 65   }
```

MPS-07 | Return Value Not Handled

Category	Severity	Location	Status
Volatile Code	● Informational	presale/MoniPreSale2.sol: 155	ⓘ Acknowledged

Description

The return value of `transferPresale2()` is not properly handled.

```
155         erc20Contract.transferPresale2(depositor, contributedAmount.mul(amount));
```

`transferPresale2()` is not void-return functions per `IErc20Contract` interface. Ignoring the return value of the function might cause some unexpected exceptions, especially if the called function does not revert automatically on failure.

Recommendation

We recommend checking the return value of the aforementioned function and handling both success and failure cases based on the business logic.

MPS-08 | Missing Error Messages

Category	Severity	Location	Status
Coding Style	● Informational	presale/MoniPreSale2.sol: 43	ⓘ Acknowledged

Description

The **require** can be used to check for conditions and throw an exception if the condition is not met. It is better to provide a string message containing details about the error that will be passed back to the caller.

Recommendation

We recommend adding appropriate error messages in the aforementioned **require** statements.

MTC-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	MoniToken.sol: 584, 592, 598, 744, 748, 752, 756	Partially Resolved

Description

In the contract `MoniToken`, the role `admin` has the authority over the following function:

- `MoniToken.transferOwnership()` to transfer the `admin` role;
- `MoniToken.setPreSale1ContractNotYetSet()` to set the first pre-sale contract;
- `MoniToken.setPreSale2ContractNotYetSet()` to set the second pre-sale contract;
- `MoniToken.pause()` to pause the contract;
- `MoniToken.unpause()` to unpause the contract;
- `MoniToken.pausedAddress()` to pause certain address;
- `MoniToken.unPausedAddress()` to unpause certain address.

Any compromise to the `admin` account may allow the hacker to take advantage of this and manipulate the protocol.

Recommendation

We advise the client to carefully manage the `admin` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Indicatively, here is some feasible suggestions that would also mitigate the potential risk at the different level in term of short-term and long-term:

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

The Monsta Team applies a Gnosis multisig wallet 0x9573c88aE3e37508f87649f87c4dd5373C9F31e0 for the contract admin. It requires 2 (out of 3) team members to confirm every transaction in order to execute it, which helps prevent unauthorized access to executing the aforementioned functions.

In addition, we recommend any plan to invoke the aforementioned function should be also considered to move to the execution queue of the `Timelock` contract and dynamic runtime updates in the project should be notified to the community ahead.

MTC-02 | Missing Error Messages

Category	Severity	Location	Status
Coding Style	● Minor	MoniToken.sol: 549, 554, 559, 564, 569	ⓘ Acknowledged

Description

The **require** can be used to check for conditions and throw an exception if the condition is not met. It is better to provide a string message containing details about the error that will be passed back to the caller.

Recommendation

We recommend adding appropriate error messages in the aforementioned **require** statements.

MTC-03 | Variable Could Be Declared as `constant`

Category	Severity	Location	Status
Gas Optimization, Coding Style	● Informational	MoniToken.sol: 41	ⓘ Acknowledged

Description

The variable `_totalSupply` could be declared as `constant` since these state variables are never to be changed.

Recommendation

We recommend declaring `_totalSupply` as `constant`.

MTC-04 | Missing Event Emissions for Significant Transactions

Category	Severity	Location	Status
Coding Style	● Informational	MoniToken.sol: 584, 592, 598, 744, 748, 752, 756	ⓘ Acknowledged

Description

The function that affects the status of sensitive variables should be able to emit events as notifications to users. For example,

- `MoniToken.transferOwnership()` to transfer the `admin` role;
- `MoniToken.setPreSale1ContractNotYetSet()` to set the first pre-sale contract;
- `MoniToken.setPreSale2ContractNotYetSet()` to set the second pre-sale contract;
- `MoniToken.pause()` to pause the contract;
- `MoniToken.unpause()` to unpause the contract;
- `MoniToken.pausedAddress()` to pause certain address;
- `MoniToken.unPausedAddress()` to unpause certain address.

Recommendation

We recommend emitting events for all the essential state variables that are possible to be changed during the runtime.

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you (“Customer” or the “Company”) in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK’s prior written consent in each instance.

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED “AS IS” AND “AS

AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING

MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.