Num ARS

# Smart Contract Audit Report

SYSFIXED

23 Oct 2021

# TABLE OF CONTENTS

# AUDITED DETAILS

## Audited Project

| Project name | Token ticker | Blockchain |
|---|---|---|
| Num ARS | nuARS | Binance Smart Chain |

## Addresses

| | |
|---|---|
| **Contract address** | 0x91bc956f064d755db2e4efe839ef0131e0b07e28 |
| **Contract deployer address** | 0xAaf62F834F942BF0C5cC21DA67A8C7cb183f8C18 |

## Project Website

**https://num.finance/**

## Codebase

**https://bscscan.com/address/0x91bc956f064d755db2e4efe839ef0131e0b07e28#code**

# SUMMARY

NUM Finance is a project that aims to create stable cryptocurrencies that track the value of emerging market currencies. These cryptocurrencies are called Num Stablecoins.

## Contract Summary

**Documentation Quality**

Num ARS provides a very good documentation with standard of solidity base code.

- The technical description is provided clearly and structured and also dont have any high risk issue.

**Code Quality**

The Overall quality of the basecode is standard.

- Standard solidity basecode and rules are already followed by Num ARS with the discovery of several low issues.

**Test Coverage**

Test coverage of the project is 100% ( Through Codebase )

## Audit Findings Summary

- SWC-103 | Pragma statements can be allowed to float when a contract is intended on lines 11, 37, 116, 332, 638, 691, 779, 819, 929, 1007, 1306, 1497 and 1714.

# CONCLUSION

We have audited the Num ARS project released on October 2021 to discover issues and identify potential security vulnerabilities in Num ARS Project. This process is used to find technical issues and security loopholes which might be found in the smart contract.

The security audit report provides satisfactory results with low-risk issues.

The issues found in the Num ARS smart contract code do not pose a considerable risk. The writing of the contract is close to the standard of writing contracts in general. The low-risk issues found are some floating pragma is set. Specifying a fixed compiler version is recommended to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

# AUDIT RESULT

| Article | Category | Description | Result |
|---|---|---|---|
| Default Visibility | SWC-100<br>SWC-108 | Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously. | PASS |
| Integer Overflow and Underflow | SWC-101 | If unchecked math is used, all math operations should be safe from overflows and underflows. | PASS |
| Outdated Compiler Version | SWC-102 | It is recommended to use a recent version of the Solidity compiler. | PASS |
| Floating Pragma | SWC-103 | Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly. | ISSUE FOUND |
| Unchecked Call Return Value | SWC-104 | The return value of a message call should be checked. | PASS |
| Unprotected Ether Withdrawal | SWC-105 | Due to missing or insufficient access controls, malicious parties can withdraw from the contract. | PASS |
| SELFDESTRUCT Instruction | SWC-106 | The contract should not be self-destructible while it has funds belonging to users. | PASS |
| Reentrancy | SWC-107 | Check effect interaction pattern should be followed if the code performs recursive call. | PASS |
| Uninitialized Storage Pointer | SWC-109 | Uninitialized local storage variables can point to unexpected storage locations in the contract. | PASS |
| Assert Violation | SWC-110<br>SWC-123 | Properly functioning code should never reach a failing assert statement. | PASS |
| Deprecated Solidity Functions | SWC-111 | Deprecated built-in functions should never be used. | PASS |
| Delegate call to Untrusted Callee | SWC-112 | Delegatecalls should only be allowed to trusted addresses. | PASS |

| | | | |
|---|---|---|---|
| DoS (Denial of Service) | SWC-113<br>SWC-128 | Execution of the code should never be blocked by a specific contract state unless required. | PASS |
| Race Conditions | SWC-114 | Race Conditions and Transactions Order Dependency should not be possible. | PASS |
| Authorization through tx.origin | SWC-115 | tx.origin should not be used for authorization. | PASS |
| Block values as a proxy for time | SWC-116 | Block numbers should not be used for time calculations. | PASS |
| Signature Unique ID | SWC-117<br>SWC-121<br>SWC-122 | Signed messages should always have a unique id. A transaction hash should not be used as a unique id. | PASS |
| Incorrect Constructor Name | SWC-118 | Constructors are special functions that are called only once during the contract creation. | PASS |
| Shadowing State Variable | SWC-119 | State variables should not be shadowed. | PASS |
| Weak Sources of Randomness | SWC-120 | Random values should never be generated from Chain Attributes or be predictable. | PASS |
| Write to Arbitrary Storage Location | SWC-124 | The contract is responsible for ensuring that only authorized user or contract accounts may write to sensitive storage locations. | PASS |
| Incorrect Inheritance Order | SWC-125 | When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order. The rule of thumb is to inherit contracts from more /general/ to more /specific/. | PASS |
| Insufficient Gas Griefing | SWC-126 | Insufficient gas griefing attacks can be performed on contracts which accept data and use it in a sub-call on another contract. | PASS |
| Arbitrary Jump Function | SWC-127 | As Solidity doesnt support pointer arithmetics, it is impossible to change such variable to an arbitrary value. | PASS |

| | | | |
|---|---|---|---|
| Typographical Error | SWC-129 | A typographical error can occur for example when the intent of a defined operation is to sum a number to a variable. | PASS |
| Override control character | SWC-130 | Malicious actors can use the Right-To-Left-Override unicode character to force RTL text rendering and confuse users as to the real intent of a contract. | PASS |
| Unused variables | SWC-131 SWC-135 | Unused variables are allowed in Solidity and they do not pose a direct security issue. | PASS |
| Unexpected Ether balance | SWC-132 | Contracts can behave erroneously when they strictly assume a specific Ether balance. | PASS |
| Hash Collisions Variable | SWC-133 | Using abi.encodePacked() with multiple variable length arguments can, in certain situations, lead to a hash collision. | PASS |
| Hardcoded gas amount | SWC-134 | The transfer() and send() functions forward a fixed amount of 2300 gas. | PASS |
| Unencrypted Private Data | SWC-136 | It is a common misconception that private type variables cannot be read. | PASS |

# SMART CONTRACT ANALYSIS

| | |
|---|---|
| **Started** | Friday Oct 22 2021 09:01:58 GMT+0000 (Coordinated Universal Time) |
| **Finished** | Saturday Oct 23 2021 04:00:45 GMT+0000 (Coordinated Universal Time) |
| **Mode** | Standard |
| **Main Source File** | Token.sol |

## Detected Issues

| ID | Title | Severity | Status |
|---|---|---|---|
| SWC-103 | A FLOATING PRAGMA IS SET. | low | acknowledged |
| SWC-103 | A FLOATING PRAGMA IS SET. | low | acknowledged |
| SWC-103 | A FLOATING PRAGMA IS SET. | low | acknowledged |
| SWC-103 | A FLOATING PRAGMA IS SET. | low | acknowledged |
| SWC-103 | A FLOATING PRAGMA IS SET. | low | acknowledged |
| SWC-103 | A FLOATING PRAGMA IS SET. | low | acknowledged |
| SWC-103 | A FLOATING PRAGMA IS SET. | low | acknowledged |
| SWC-103 | A FLOATING PRAGMA IS SET. | low | acknowledged |
| SWC-103 | A FLOATING PRAGMA IS SET. | low | acknowledged |
| SWC-103 | A FLOATING PRAGMA IS SET. | low | acknowledged |
| SWC-103 | A FLOATING PRAGMA IS SET. | low | acknowledged |
| SWC-103 | A FLOATING PRAGMA IS SET. | low | acknowledged |
| SWC-103 | A FLOATING PRAGMA IS SET. | low | acknowledged |

# SWC-103 | A FLOATING PRAGMA IS SET.

LINE 11

## low SEVERITY

The current pragma Solidity directive is ""=0.6.0<0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

## Source File

- Token.sol

## Locations

```
10
11   pragma solidity >=0.6.0 <0.8.0;
12
13   /*
14   * @dev Provides information about the current execution context, including the
15
```

# SWC-103 | A FLOATING PRAGMA IS SET.

LINE 37

## low SEVERITY

The current pragma Solidity directive is ""=0.6.0<0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

## Source File

- Token.sol

## Locations

```
36
37   pragma solidity >=0.6.0 <0.8.0;
38
39   /**
40   * @dev Interface of the ERC20 standard as defined in the EIP.
41
```

# SWC-103 | A FLOATING PRAGMA IS SET.

LINE 116

## low SEVERITY

The current pragma Solidity directive is ""=0.6.0<0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

## Source File

- Token.sol

## Locations

```
115
116   pragma solidity >=0.6.0 <0.8.0;
117
118   /**
119   * @dev Wrappers over Solidity's arithmetic operations with added overflow
120
```

# SWC-103 | A FLOATING PRAGMA IS SET.

LINE 332

## low SEVERITY

The current pragma Solidity directive is ""=0.6.0<0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

## Source File

- Token.sol

## Locations

```
331
332   pragma solidity >=0.6.0 <0.8.0;
333
334
335
336
```

# SWC-103 | A FLOATING PRAGMA IS SET.

LINE 638

## low SEVERITY

The current pragma Solidity directive is ""=0.6.0<0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

## Source File

- Token.sol

## Locations

```
637
638   pragma solidity >=0.6.0 <0.8.0;
639
640   /**
641   * @dev Interface of the ERC20 Permit extension allowing approvals to be made via
signatures, as defined in
642
```

# SWC-103 | A FLOATING PRAGMA IS SET.

LINE 691

## low SEVERITY

The current pragma Solidity directive is ""=0.6.0<0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

## Source File

- Token.sol

## Locations

```
690
691   pragma solidity >=0.6.0 <0.8.0;
692
693   /**
694   * @dev Elliptic Curve Digital Signature Algorithm (ECDSA) operations.
695
```

# SWC-103 | A FLOATING PRAGMA IS SET.

LINE 779

## low SEVERITY

The current pragma Solidity directive is ""=0.6.0<0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

## Source File

- Token.sol

## Locations

```
778
779   pragma solidity >=0.6.0 <0.8.0;
780
781   /**
782   * @title Counters
783
```

# SWC-103 | A FLOATING PRAGMA IS SET.

LINE 819

## low SEVERITY

The current pragma Solidity directive is ""=0.6.0<0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

## Source File

- Token.sol

## Locations

```
818
819   pragma solidity >=0.6.0 <0.8.0;
820
821   /**
822   * @dev https://eips.ethereum.org/EIPS/eip-712[EIP 712] is a standard for hashing
and signing of typed structured data.
823
```

# SWC-103 | A FLOATING PRAGMA IS SET.

LINE 929

## low SEVERITY

The current pragma Solidity directive is ""=0.6.5<0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

## Source File

- Token.sol

## Locations

```
928
929   pragma solidity >=0.6.5 <0.8.0;
930
931
932
933
```

# SWC-103 | A FLOATING PRAGMA IS SET.

LINE 1007

## low SEVERITY

The current pragma Solidity directive is ""=0.6.0<0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

## Source File

- Token.sol

## Locations

```
1006
1007   pragma solidity >=0.6.0 <0.8.0;
1008
1009   /**
1010   * @dev Library for managing
1011
```

# SWC-103 | A FLOATING PRAGMA IS SET.

LINE 1306

## low SEVERITY

The current pragma Solidity directive is ""=0.6.2<0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

## Source File

- Token.sol

## Locations

```
1305
1306   pragma solidity >=0.6.2 <0.8.0;
1307
1308   /**
1309   * @dev Collection of functions related to the address type
1310
```

# SWC-103 | A FLOATING PRAGMA IS SET.

LINE 1497

## low SEVERITY

The current pragma Solidity directive is ""=0.6.0<0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

## Source File

- Token.sol

## Locations

```
1496
1497   pragma solidity >=0.6.0 <0.8.0;
1498
1499
1500
1501
```

# SWC-103 | A FLOATING PRAGMA IS SET.

LINE 1714

## low SEVERITY

The current pragma Solidity directive is ""^0.7.3"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

## Source File

- Token.sol

## Locations

```
1713
1714   pragma solidity ^0.7.3;
1715
1716
1717
1718
```

# 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 Sysfixed'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 Sysfixed 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 is a limited report on our findings based on our analysis, in accordance with good industry practice as of the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the below disclaimer below – please make sure to read it in full.

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.

This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Sysfixed and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (Sysfixed) owe no duty of care.

# ABOUT US

Sysfixed is a blockchain security certification organization established in 2021 with the objective to provide smart contract security services and verify their correctness in blockchain-based protocols. Sysfixed automatically scans for security vulnerabilities in Ethereum and other EVM-based blockchain smart contracts. Sysfixed a comprehensive range of analysis techniques—including static analysis, dynamic analysis, and symbolic execution—can accurately detect security vulnerabilities to provide an in-depth analysis report. With a vibrant ecosystem of world-class integration partners that amplify developer productivity, Sysfixed can be utilized in all phases of your project's lifecycle. Our team of security experts is dedicated to the research and improvement of our tools and techniques used to fortify your code.