DetectBox
Powered by UNSNARL

#DECENTRALISED PROACTIVE AUDIT WAR-ROOMS

Token Contract Security Assessment

ETHER-VERSE

JUNE 13TH, 2023

TABLE OF CONTENTS

SUMMARY

This report has been prepared for Ether Verse LLC to discover issues and vulnerabilities in the source code of the ICO TOKEN 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

CERTIFICATION

This is to Certify that the **Ether-Verse Security Report** was prepared using gold standard web3 security standards with all standard and advanced checks deployed.

The **Main Auditor** Involved in the Audit was -
Samrat Gupta(@Sm4rty_)

The **Detect-Warden** involved in the Audit was -
Rohan Jha(@rohan16___)

The Auditor mentioned hereby was chosen after self due-dilligence from the project's end.

All reported issues were either acknowledged or fixed by the project.

DetectBox's audit mitigation & residual vulnerability check did not find any major vulnerability to report within the Scope and requirements of the audit.

From
Team DetectBox

Ether-Verse
Security Assessment Report

ABOUT DETECTBOX

DetectBox by **UNSNARL** brings to you world's first Decentralised Audit War-Rooms. An intense collaborative Environment where chosen auditors/audit team, project's decision makers, Projects developers and Detect-Wardens join hands to conduct rigorous security audits with absolute transparency followed by a **Proof of Audit (POA).**

We at DetectBox not only find bugs in smart-contracts but also take a deep look into the project's **tokenomics**, **white-paper**, **business logic**, **functional flows** and overall development health.

At DetectBox, you get to choose from a pool of Independent Security researchers which is curated by strong due-diligence. List your project requirements, verify the auditor's past work from their profiles and choose the auditor that rightly fits your project's needs and your budget.

Our auditors have successfully completed **200+ audits**, found over **2200+ vulnerabilities** and secured over **$102M+ worth of TVL.**

To know more about us visit - detectbox.io
To see our docs - https://unsnarl.gitbook.io/detectbox/

Yours Securely
UNSNARL

OVERVIEW

Project Summary

The MDU Token is an ERC20 token with a total supply of 1 trillion tokens. The price of each token has been hardcoded to 0.00025 Ether. Users can invest in the token using the Presale contract.

Audit Summary

Delivery Date : 13 June 2023
Audit Methodology : Static Analysis, Manual Review

Vulnerability Summary

Severity classification

Severity	Impact : High	Impact : Medium	Impact : Low
Likelihood : High	Critical	High	Medium
Likelihood : Medium	High	Medium	Low
Likelihood : Low	Medium	Low	Low

Findings Summary

High	0 issues
Medium	1 issue
Low	3 issues
Informational	2 issues
Gas Optimisations	4 issues

OVERVIEW

Audit Scope

The code under review is composed of a single smart contract and interfaces written in the Solidity programming language and includes 149 nSLOC- normalized source lines of code (only source-code lines; no comments, no blank lines).

TypeFile	Contracts	Interfaces	Lines	nSLOC
MDUToken.sol	1	328	267	149

Auditors Involved :

Main Auditor -

Samrat Gupta - https://app.detectbox.io/profile/sm4rty

Detect Warden -

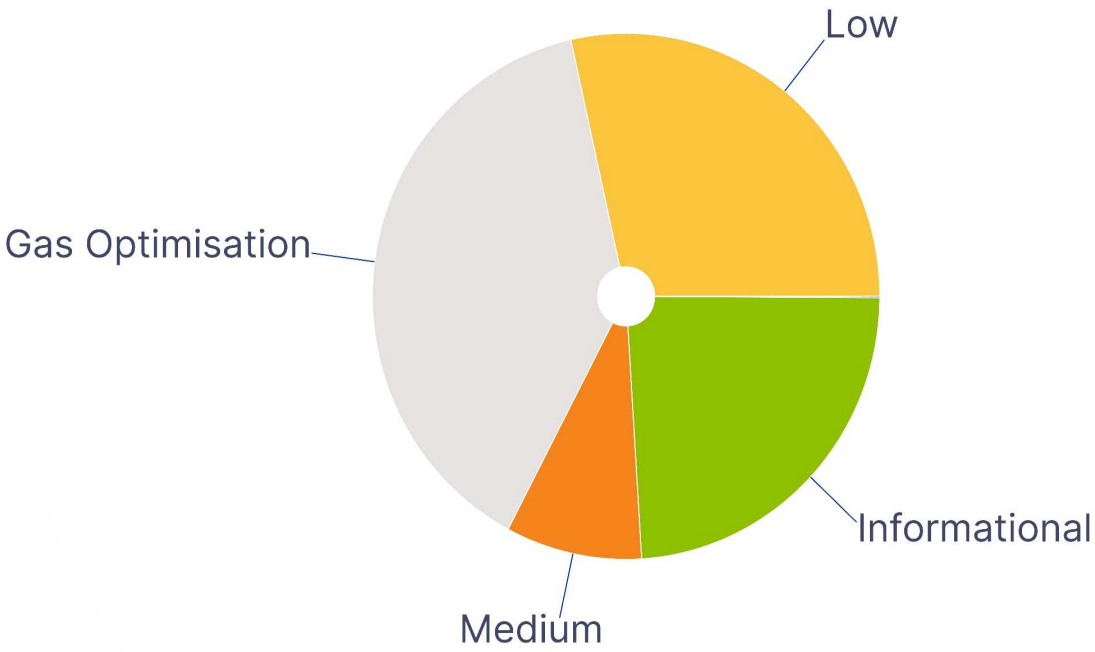
Rohan Jha - https://app.detectbox.io/profile/Rohan16

FINDINGS

Detailed Summary of Findings

Sl. No.	Name	Severity
1.	Transfer ETH by using transfer() may cause this transaction to fail.	Medium
2.	Avoiding the use of floating Pragma.	Low
3.	Lack of 2-step transfer of ownership.	Low
4.	Lack of 0-address check.	Low
5.	Use the latest version of solidity.	Informational
6.	Contracts should have full test coverage.	Informational
7.	Lack of 0-address Using > 0 costs more gas than != 0 when used on a uint in a require() statement.	Gas Optimization
8.	Splitting require() statements that use && saves gas	Gas Optimization
9.	Custom Errors instead of Revert Strings to save Gas	Gas Optimization
10.	Strict inequalities (>) are more expensive than non-strict ones (>=)	Gas Optimization

FINDINGS



Static Analysis

No major issues were found. Some false positive errors were reported by the tool. All the other issues have been categorized above according to their level of severity.

Manual Review

High Severity Issues

No High Severity issues were found.

FINDINGS

Medium Severity Issues

M-01. Transfer ETH by using transfer() may cause this transaction to fail

Transferring ETH by using **transfer()** may cause this transaction to fail. Due to the fact that .transfer() and .send() forward exactly 2,300 gas to the recipient. This hardcoded gas stipend aimed to prevent reentrancy vulnerabilities, but this only makes sense under the assumption that gas costs are constant. Recently EIP 1884 was included in the Istanbul hard fork. One of the changes included in EIP 1884 is an increase in the gas cost of the SLOAD operation, causing a contract's fallback function to cost more than 2300 gas.

Instances:

File: MDUToken.sol:301:

```
function Investingtoken() payable public returns(bool success) {
           uint256 tokens =  msg.value / tokenPrice;
          _balances[_msgSender()] += tokens;
          _balances[_creator] -= tokens;
          recipient.transfer(msg.value); //@audit Use call for transferring eth
instead of transfer.
          receivedfund += msg.value;
          return true;

      }
```

FINDINGS

Recommended Mitigation Steps:

Use **call{value: msg.value}()** instead of **transfer()** to send Ether.

```
(bool success, ) = payable(recipient).call{value: msg.value}("");
require(success, "call failed");
```

Low Severity Issues

L-01. Avoiding the use of floating Pragma.

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Instances:

File: MDUToken.sol:3:

```
pragma solidity ^0.8.15;
```

Recommended Mitigation Steps:

Avoid Floating pragma and use fixed solidity version

FINDINGS

L-02. Lack of 2-step transfer of ownership.

Ownable2Step is safer than Ownable for smart contracts because the owner cannot accidentally transfer smart contract ownership to a mistyped address. Rather than directly transferring to the new owner, the transfer only completes when the new owner accepts ownership. Also, If the nominated EOA account is not a valid account, it is entirely possible the owner may accidentally transfer ownership to an uncontrolled account, breaking all functions with the **onlyOwner()** modifier.

Instances:

File: MDUToken.sol:81:

```
function transferOwnership(address newOwner) public virtual onlyOwner {
        require(
            newOwner != address(0),
            "Ownable: new owner is the zero address"
        );
        _transferOwnership(newOwner);
 }

function _transferOwnership(address newOwner) internal virtual {
        address oldOwner = _owner;
        _owner = newOwner;
        emit OwnershipTransferred(oldOwner, newOwner);
    }
}
```

FINDINGS

Recommended Mitigation Steps:

Recommend considering implementing a two step process where the owner nominates an account and the nominated account needs to call an **acceptOwnership()** function for the transfer of ownership to fully succeed. This ensures the nominated EOA account is a valid and active account.

L-03. Lack of 0-address check.

Checking addresses against zero-address during initialization or during setting is a security best practice. However, such checks are missing in the constructor of the Presale contract during recipient address initializations.

Instances:

File: MDUToken.sol:288:

```
constructor(address payable _recipient)  {
          admin = _msgSender();
          recipient = _recipient; //@audit No check for Zero-address for recipie
nt.
      }
```

Recommended Mitigation Steps:

Recommend adding zero-address checks for all initializations of address state variables.

FINDINGS

Informational Issues

I-01. Use the latest version of solidity.

When deploying contracts, you should use the latest released version of Solidity. Apart from exceptional cases, only the latest version receives security fixes. Furthermore, breaking changes, as well as new features, are introduced regularly.

Instances:

File: MDUToken.sol:3:

```
pragma solidity ^0.8.15;
```

Recommended Mitigation Steps:

Use the latest version of solidity i.e. 0.8.19 or 0.8.20.

I-02. Contracts should have full test coverage.

While 100% code coverage does not guarantee that there are no bugs, it often will catch easy-to-find bugs, and will ensure that there are fewer regressions when the code invariably has to be modified. Furthermore, in order to get full coverage, code authors will often have to re-organize their code so that it is more modular, so that each component can be tested separately, which reduces interdependencies between modules and layers, and makes for code that is easier to reason about and audit. Contracts should have 90%+ code coverage.

FINDINGS

References:

https://gist.github.com/CloudEllie/6639dbfd7dc1809a3baa28bb2895e1d9#n31-contracts-should-have-full-test-coverage

Gas Optimisation Issues

G-01. Using > 0 costs more gas than != 0 when used on a uint in a require() statement.

0 is less efficient than != 0 for unsigned integers (with proof) != 0 costs less gas compared to > 0 for unsigned integers in require statements with the optimizer enabled (6 gas) Proof: While it may seem that > 0 is cheaper than !=, this is only true without the optimizer enabled and outside a require statement. If you enable the optimizer at 10k AND you're in a require statement, this will save gas. You can see this tweet for more proof: https://twitter.com/gzeon/status/1485428085885640706

Instances:

```
MDUToken.sol:206:        require(_value > 0 && _balances[_msgSender()] >= _valu
e);
MDUToken.sol:219:        require(_value > 0 && _balances[_msgSender()] >= _valu
e);
```

FINDINGS

Remediation:

I suggest changing > 0 with != 0. Also, please enable the Optimizer.

G-02. Splitting require() statements that use && saves gas.

Require statements including conditions with the && operator can be broken down in multiple require statements to save gas.

Instances:

```
MDUToken.sol:206:        require(_value > 0 && _balances[_msgSender()] >= _valu
e);
MDUToken.sol:219:        require(_value > 0 && _balances[_msgSender()] >= _valu
e);
```

Mitigation:

Breakdown each condition in a separate require statement (though require statements should be replaced with custom errors)

G-03. Custom Errors instead of Revert Strings to save Gas.

Custom errors from Solidity 0.8.4 are cheaper than revert strings (cheaper deployment cost and runtime cost when the revert condition is met). Starting from Solidity v0.8.4,there is a convenient and gas-efficient way to explain to users why an operation failed through the use of custom errors.

FINDINGS

Until now, you could already use strings to give more information about failures (e.g., revert("Insufficient funds.");),but they are rather expensive, especially when it comes to deploy cost, and it is difficult to use dynamic information in them. Custom errors are defined using the error statement, which can be used inside and outside of contracts (including interfaces and libraries).

Instances:

```
MDUToken.sol:51:        require(owner() == _msgSender(), "Ownable: caller is not
the owner");
```

Remediation:

I suggest replacing revert strings with custom errors.

G-04. Strict inequalities (>) are more expensive than non-strict ones (>=).

Strict inequalities (>) are more expensive than non-strict ones (>=). This is due to some supplementary checks (ISZERO, 3 gas. I suggest using >= instead of > to avoid some opcodes here:

Instances:

```
MDUToken.sol:234:                _value > 0 &&
```

References:

https://code4rena.com/reports/2022-04-badger-citadel/#g-31--is-cheaper-than

APPENDIX

Finding Categories

Finding Categories

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.

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.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

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.

DISCLAIMER

DetectBox (by UNSNARL) has conducted an independent security review to verify the integrity of and highlight any vulnerabilities, bugs or errors, intentional or unintentional, that may be present in the codebase that were provided for the scope of this Engagement.

Blockchain technology and the cryptographic assets it supports are nascent technologies. This makes them extremely volatile assets. Any assessment report obtained on such volatile and nascent assets may include unpredictable results which may lead to positive or negative outcomes.

In some cases, services provided may be reliant on a variety of third parties. This security review does not constitute endorsement, agreement or acceptance for the Project and technology that was reviewed. Users relying on this security review should not consider this as having any merit for financial advice or technological due diligence in any shape, form or nature.

The veracity and accuracy of the findings presented in this report relate solely to the proficiency, competence, aptitude and discretion of our auditors. UNSNARL and its employees make no guarantees, nor assurance that the contracts are free of exploits, bugs, vulnerabilities, deprecation of technologies or any system / economical / mathematical malfunction.

DISCLAIMER

All the information/opinions/suggestions provided in this report does not constitute financial or investment advice, nor should it be used to signal that any person reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report.

Information in this report is provided 'as is'. UNSNARL is under no covenant to the completeness, accuracy or solidity of the contracts reviewed. UNSNARL's goal is to help reduce the attack vectors/surface and the high level of variance associated with utilizing new and consistently changing technologies.

UNSNARL in no way claims any guarantee, warranty or assurance of security or functionality of the technology that was in scope for this security review.

In no event will UNSNARL, its partners, employees, agents or any parties related to the design/creation of this security review be ever liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this security review.

Cryptocurrencies and all other technologies directly or indirectly related to cryptocurrencies are not standardized, highly prone to malfunction and extremely speculative by nature. No due diligence and/or safeguards may be insufficient and users should exercise maximum caution when participating and/or investing in this nascent industry.

DISCLAIMER

The preparation of this security review has made all reasonable attempts to provide clear and actionable recommendations to the Project team (the “client”) with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts in scope for this engagement.

It is the sole responsibility of the Project team to provide adequate levels of test and perform the necessary checks to ensure that the contracts are functioning as intended, and more specifically to ensure that the functions contained within the contracts in scope have the desired intended effects, functionalities and outcomes, as documented by the Project team.

All services, the security reports, discussions, work product, attack vectors description or any other materials, products or results of this security review engagement is provided "as is" and "as available" and with all faults, uncertainty and defects without warranty or guarantee of any kind.

UNSNARL will assume no liability or responsibility for delays, errors, mistakes, or any inaccuracies of content, suggestions, materials or for any loss, delay, damage of any kind which arose as a result of this engagement/security review.

UNSNARL will assume no liability or responsibility for any personal injury, property damage, of any kind whatsoever that resulted in this engagement and the customer having access to or use of the products, engineers, services, security report, or any other other materials.

For avoidance of doubt, this report, its content, access, and/or usage thereof, including any associated services or materials, shall not be considered or relied upon as any form of financial, investment, tax, legal, regulatory, or any other type of advice.

GLORY OF AUDITORS AT DETECTBOX

DetectBox has created a pool of best auditors across the globe with significant experience in the web3 security industry auditing multiple protocols across multiple chains. Making audits better through **Proof of Talent**.

217+

Projects Audited

$100M+

Amount Secured

2200+

Vulnerabilities Detected

Follow Our Journey on

@unsnarl_secure

UNSNARL

founders@unsnarl.io

www.detectbox.io