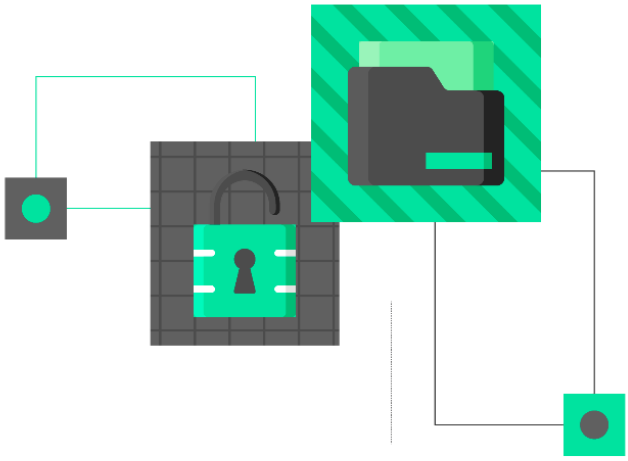
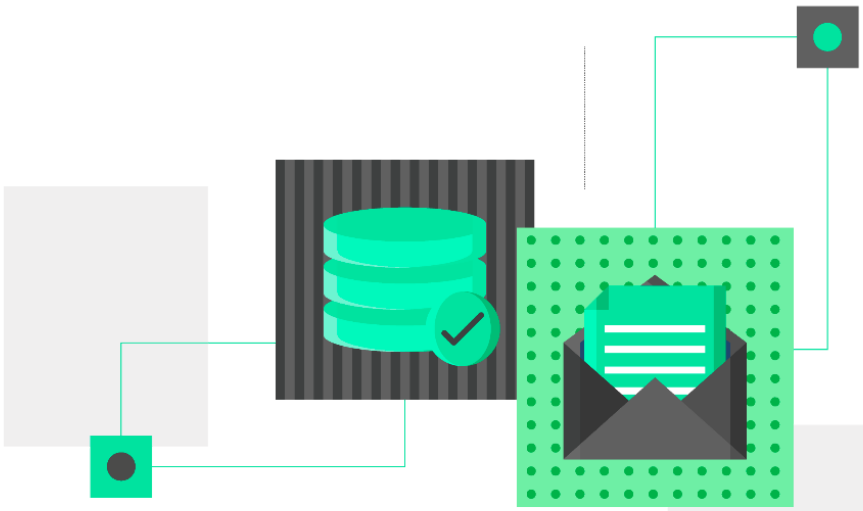
Mantisec Labs

Smart Contract Audit



Presearch

PreTokenv3.sol



Oct 2023

Contents

Disclaimer

This disclaimer is to inform you that the report you are reading has been prepared by Mantisec Labs for informational purposes only and should not be considered as investment advice. It is important to conduct your own independent investigation before making any decisions based on the information contained in the report. The report is provided "as is" without any warranties or conditions of any kind and Mantisec Labs excludes all representations, warranties, conditions, and other terms. Additionally, Mantisec Labs assumes no liability or responsibility for any kind of loss or damage that may result from the use of this report.

It is important to note that the analysis in the report is limited to the security of the smart contracts only and no applications or operations were reviewed. The report contains proprietary information, and Mantisec Labs holds the copyright to the text, images, photographs, and other content. If you choose to share or use any part of the report, you must provide a direct link to the original document and credit Mantisec Labs as the author.

By reading this report, you are accepting the terms of this disclaimer. If you do not agree with these terms, it is advisable to discontinue reading the report and delete any copies in your possession.

Audit Process & Methodology

The Mantisec Labs team carried out a thorough audit for the project, starting with an in-depth analysis of code design patterns. This initial step ensured the smart contract's architecture was well-structured and securely integrated with third-party smart contracts and libraries.Also, our team conducted a thorough line-by-line inspection of the smart contract,seeking out potential issues such as Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, among others.
During the Unit testing phase, we assessed the functions authored by the developer to ascertain their precise functionality. Our Automated Testing procedures leveraged proprietary tools designed in-house to spot vulnerabilities and security flaws within the Smart Contract. The code was subjected to an in-depth audit administered by an independent team of auditors, encompassing the following critical aspects:

- Scrutiny of the smart contract's structural analysis to verify its integrity.
- Extensive automated testing of the contract
- A manual line-by-line Code review, undertaken with the aim of evaluating, analyzing, and identifying potential security risks.
- An evaluation of the contract's intended behavior, encompassing a review of provided documentation to ensure the contract conformed to expectations.
- Rigorous verification of storage layout in upgradeable contracts.
- An integral component of the audit procedure involved the identification and recommendation of enhanced gas optimization techniques for the contract

Audit Purpose

Mantisec Labs was hired by the Presearch team to review their smart contract.
This audit was conducted in **October 2023**.

The main reasons for this review were:

- To find any possible security issues in the smart contract.
- To carefully check the logic behind the given smart contract.

This report provides valuable information for assessing the level of risk associated with this smart contract and offers suggestions on how to improve its security by addressing any identified issues.

Contract Details

Project Name	Presearch
Contract link	https://github.com/PresearchOfficial/PRE-Token/blob/main/contracts/PRETokenV3.sol
Language	Solidity
Type	ERC20

Security Level Reference

Each problem identified in this report has been categorized into one of the following severity levels:

- **High** severity issues pose significant risks and should be addressed promptly.
- **Medium** severity issues have the potential to create problems and should be on the agenda for future fixes.
- **Low** severity issues are minor concerns and warnings. While they may not require immediate action, addressing them in the future is advisable for overall improvement.

Issues	High	Medium	Low
Open	0	0	4
Closed			

Findings

Contract Name: PreTokenv3.sol

High Severity issues

No issues were found.

Medium Severity issues

No issues were found.

Low Severity Issues

L01-Pragma Version Deprecated

Solidity is now under a fast release cycle, use a more recent version of the compiler, such as version `0.8.21`.

L02-Use != 0 instead of > 0 for Unsigned Integer Comparison

Opting for the `!= 0` comparison over `> 0` is more gas-efficient when dealing with unsigned integer types. The reason behind this is that the Solidity compiler can streamline the `!= 0` comparison into a straightforward bitwise operation, whereas the `> 0` comparison necessitates an extra subtraction operation.Thus, the use of `!= 0` can enhance gas efficiency and contribute to lowering your contract's overall expenses.

Code Location:

```
./PRE-Token-main/contracts/PRETokenV3.sol::52 => if(balanceOf(address(this)) > 0)
```

L03-Do not use Deprecated Library Functions

The _setupRole method from the AccessControl library is deprecated so it should not be used.

Code Locations

```
./PRE-Token-main/contracts/PRETokenV3.sol::19 => _setupRole(MINTER_ROLE, _msgSender());

./PRE-Token-main/contracts/TransferAuthorizableERC20.sol::40 =>
_setupRole(TRANSFER_AUTHORIZER_ROLE, _msgSender());

./PRE-Token-main/contracts/ManagedEnhancedERC20.sol::26 => _setupRole(DEFAULT_ADMIN_ROLE,
_msgSender());

./PRE-Token-main/contracts/ManagedEnhancedERC20.sol::27 => _setupRole(PAUSER_ROLE, _msgSender());
```

Reference:
https://github.com/OpenZeppelin/openzeppelin-contracts/issues/3918

L04-Use immutable for OpenZeppelin AccessControl's Roles Declarations

The reason behind this is that constant variables are computed during runtime, and their values are integrated into the contract's bytecode.So, any resource-intensive operations within the constant expression, such as a call to keccak256(), will be executed each time the contract is deployed, regardless of the consistent outcome. This can lead to increased gas costs.

On the other hand, immutable variables are assessed at compile time, and their values are incorporated into the contract's bytecode as constants.Thus, any resource-intensive operations within the immutable expression are only executed once during contract compilation, and the outcome is reused during each contract deployment. This can result in reduced gas costs compared to the use of constant variables.

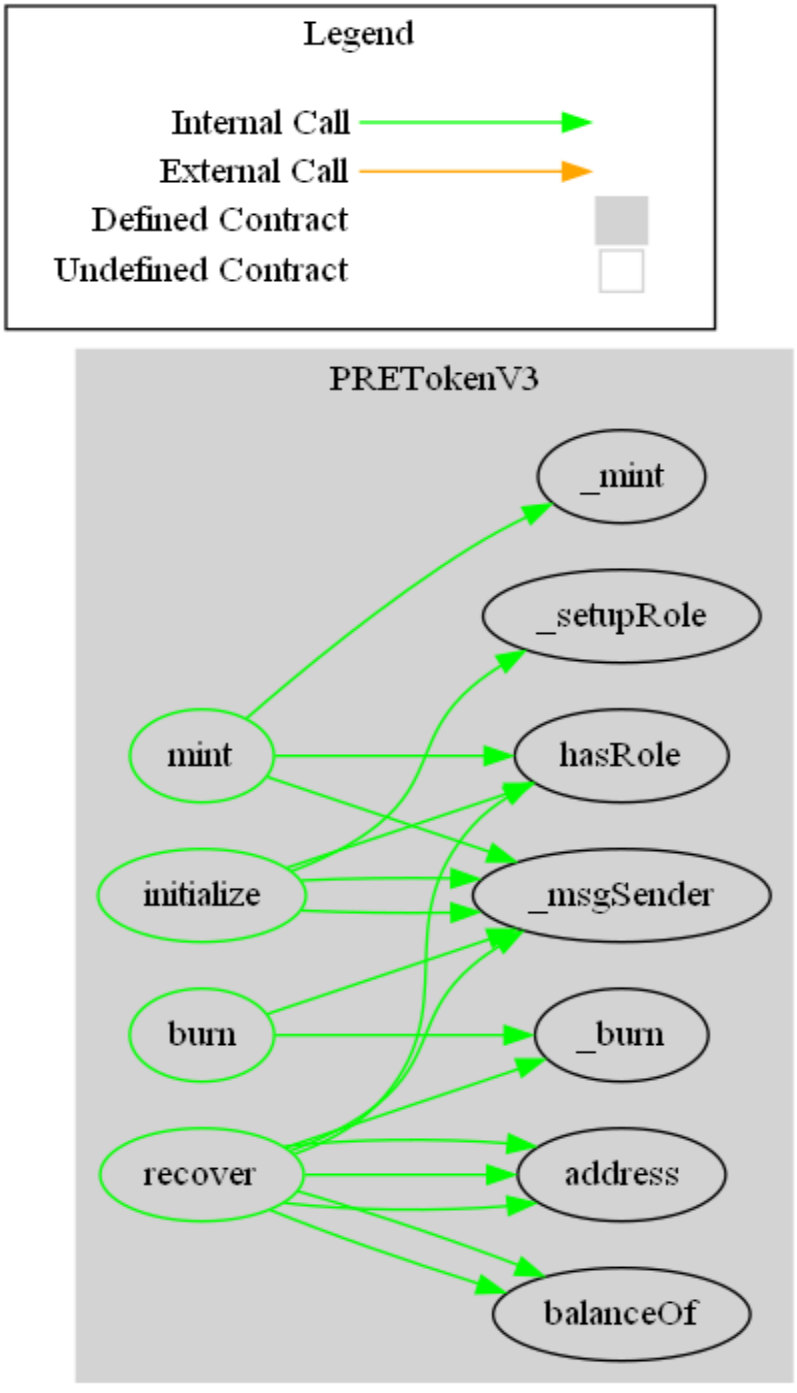
Code Location

```
./PRE-Token-main/contracts/PRETokenV3.sol::12 => bytes32 public constant MINTER_ROLE =
keccak256("MINTER_ROLE");

./PRE-Token-main/contracts/TransferAuthorizableERC20.sol::18 => bytes32 public constant
TRANSFER_AUTHORIZER_ROLE = keccak256("TRANSFER_AUTHORIZER_ROLE");

./PRE-Token-main/contracts/ManagedEnhancedERC20.sol::15 => bytes32 public constant
PAUSER_ROLE = keccak256("PAUSER_ROLE");
```

Additional Details



Concluding Remarks

To wrap it up, this PreTokenv3.sol audit has given us a good look at the contract's security and functionality.
We found some low severity issues that need attention,we suggest taking action to address these findings.