Audit Report
Deboard

January 2025

Network	Avalanche (C-Chain)
Address	0xa5EfB839A41D1b724C3CACC3bdDAED2E16373C43
Audited by	© cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Minor / Informative	ROF	Redundant Ownable Functionality	Unresolved
Minor / Informative	L09	Dead Code Elimination	Unresolved
Minor / Informative	L19	Stable Compiler Version	Unresolved

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
● Critical	Highly Likely / High Impact
● Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
● Minor / Informative	Unlikely / Low to no Impact

Review

Explorer	https://snowtrace.io/address/0xa5efb839a41d1b724c3cacc3bddaed2e16373c43

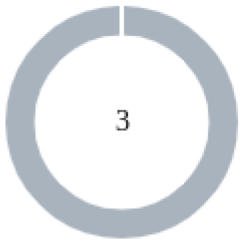
Audit Updates

Initial Audit	14 Jan 2025

Source Files

Filename	SHA256
DevaxToken.sol	6f0077b9cc8f96c6c482d67cd272cc8f0a8fb8f698b8d498e35d597f57a54a86

Findings Breakdown



Critical 0

Medium 0

Minor / Informative 3

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	3	0	0	0

ROF - Redundant Ownable Functionality

Criticality	Minor / Informative
Location	DevaxToken.sol#L7
Status	Unresolved

Description

The contract inherits from the `Ownable` abstract contract to define an owner. In smart contracts, an owner typically has elevated privileges to execute administrative functions. However, in this case, while the contract defines an owner, it does not include any administrative functionalities. Furthermore, the owner is set to `address(0)` by the deployment of the contract, effectively relinquishing any ownership or administrative control. Therefore, the inheritance of `Ownable` is redundant.

```
contract DevaxToken is ERC20, Ownable {
    constructor() ERC20("DEVAX", "DEVAX") Ownable(address(0)) {
    _mint(msg.sender, 10_000_000_000 * 10**decimals());
    }
}
```

Recommendation

Eliminating redundancies will reduce code size and enhance readability. By removing the unnecessary inheritance, the contract becomes more efficient and aids in future maintainability.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	DevaxToken.sol#L512
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function _burn(address account, uint256 amount) internal
virtual {
        require(account != address(0), "ERC20: burn from the
zero address");

        _beforeTokenTransfer(account, address(0), amount);

        uint256 accountBalance = _balances[account];
        require(accountBalance >= amount, "ERC20: burn amount
exceeds balance");
        _balances[account] = accountBalance - amount;
        _totalSupply -= amount;

        emit Transfer(account, address(0), amount);
    }
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	DevaxToken.sol#L3
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.0;
```

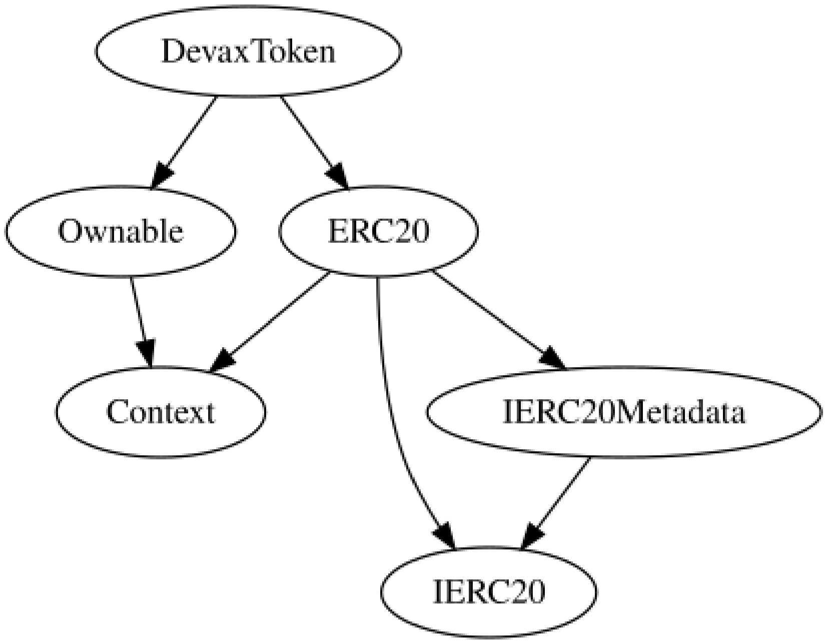
Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

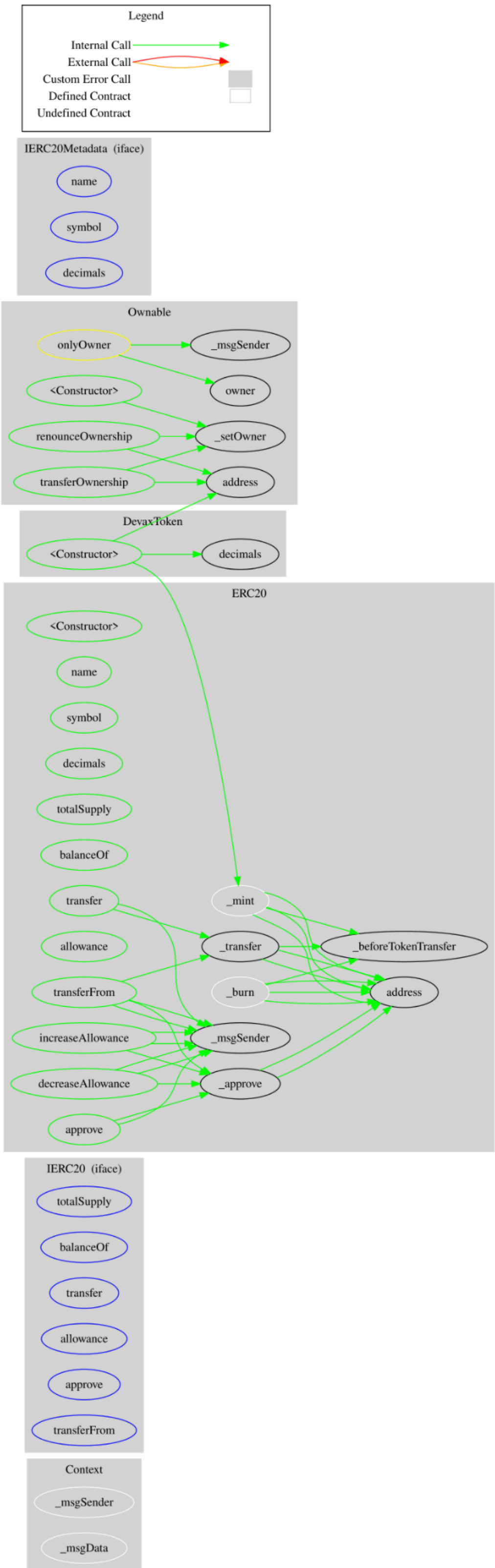
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
DevaxToken	Implementation	ERC20, Ownable		
		Public	✓	ERC20 Ownable

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
IERC20Metadata (iface)
name
symbol
decimals
Ownable
onlyOwner
_msgSender
<Constructor>
owner
renounceOwnership
_setOwner
transferOwnership
address
DevaxToken
<Constructor>
decimals
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
transfer
_mint
allowance
_transfer
_beforeTokenTransfer
transferFrom
_burn
address
increaseAllowance
_msgSender
decreaseAllowance
_approve
approve
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Context
_msgSender
_msgData

Summary

Deboard contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. Deboard is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. 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 Cyberscope's prior written consent. 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 regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document 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 represents an extensive assessment 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 Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope'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 Cyberscope are subject to dependencies and are 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.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

cyberscope.io