ExVul

SMART CONTRACT AUDIT REPORT

Nubila Token Smart Contract

OCTOBER 2025

Contents

1. EXECUTIVE SUMMARY

ExVul Web3 Security was engaged by **Nubila Token** to review smart contract implementation. The assessment was conducted in accordance with our systematic approach to evaluate potential security issues based upon customer requirement. The report provides detailed recommendations to resolve the issue and provide additional suggestions or recommendations for improvement.

The outcome of the assessment outlined in chapter 3 provides the system's owners a full description of the vulnerabilities identified, the associated risk rating for each vulnerability, and detailed recommendations that will resolve the underlying technical issue.

1.1 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [10] which is the gold standard in risk assessment using the following risk models:

- **Likelihood**: represents how likely a particular vulnerability is to be uncovered and exploited in the wild.
- **Impact**: measures the technical loss and business damage of a successful attack.
- **Severity**: determine the overall criticality of the risk.

Likelihood can be: High, Medium and Low and impact are categorized into: High, Medium, Low, Informational. Severity is determined by likelihood and impact and can be classified into five categories accordingly: Critical, High, Medium, Low, Informational shown in table 1.1.

Likelihood		IMPACT			
		Informational	**Low**	**Medium**	**High**
	High	INFO	MEDIUM	HIGH	CRITICAL
	Medium	INFO	LOW	MEDIUM	HIGH
	Low	INFO	LOW	LOW	MEDIUM

Table 1.1 Overall Risk Severity

To evaluate the risk, we will be going through a list of items, and each would be labelled with a severity category. The audit was performed with a systematic approach guided by a comprehensive assessment list carefully designed to identify known and impactful security issues. If our tool or analysis does not identify any issue, the contract can be considered safe regarding the assessed item. For any discovered issue, we might further deploy contracts on our private test environment and run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.2.

- **Basic Coding Bugs**: We first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.
- **Code and business security testing**: We further review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.
- **Additional Recommendations**: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

Category	**Assessment Item**
Basic Coding Assessment	• Apply Verification Control • Authorization Access Control • Forged Transfer Vulnerability • Forged Transfer Notification • Numeric Overflow • Transaction Rollback Attack • Transaction Block Stuffing Attack • Soft Fail Attack • Hard Fail Attack • Abnormal Memo • Abnormal Resource Consumption • Secure Random Number

Advanced Source Code Scrutiny	• Asset Security • Cryptography Security • Business Logic Review • Source Code Functional Verification • Account Authorization Control • Sensitive Information Disclosure • Circuit Breaker • Blacklist Control • System API Call Analysis • Contract Deployment Consistency Check • Abnormal Resource Consumption
Additional Recommendations	• Semantic Consistency Checks • Following Other Best Practices

Table 1.2: The Full List of Assessment Items

To better describe each issue we identified, we categorize the findings with Common Weakness Enumeration (CWE-699) [14], which is a community-developed list of software weakness types to better delineate and organize weaknesses around concepts frequently encountered in software development.

2. FINDINGS OVERVIEW

2.1 Project Info And Contract Address

Project Name	Audit Time	Language
Nubila Token	15/10/2025 - 16/10/2025	Solidity

Repository
https://github.com/Nubila/nubila-token/commits/main/

Commit Hash
9ac1a6a0e25be5d64aa5abdae80423aa4db036e7

2.2 Summary

Severity	Found
CRITICAL	0
HIGH	1
MEDIUM	1
LOW	2
INFO	1

2.3 Key Findings

Severity	Findings Title	Status
HIGH	Lack of Emergency Mechanisms in VestingManager	Fixed
MEDIUM	Function _createScheduleInternal not checking the beneficiaries is zero address	Fixed
LOW	Missing TGE Timestamp Validation	Fixed
LOW	Missing Check in updateBeneficiary Function	Fixed
INFO	Remove no use contract in the project	Fixed

Table 2.3: Key Audit Findings

3. DETAILED DESCRIPTION OF FINDINGS

3.1 Lack of Emergency Mechanisms in VestingManager

SEVERITY: HIGH **STATUS:** Fixed

PATH:

`src/VestingManager.sol`

DESCRIPTION:

The VestingManager contract lacks critical emergency mechanisms that would allow administrators to respond to security incidents. Specifically, there is no ability to pause claim functionality or recover tokens in emergency situations.

1. No Pause Mechanism: If a vulnerability is discovered in the contract or in the token it manages, there is no way to temporarily halt claims while the issue is addressed. The contract will continue to allow token distribution regardless of any discovered exploits.
2. No Token Recovery Function: If tokens are accidentally sent to the contract, or if a critical issue requires intervention, there is no mechanism to extract these tokens. They become permanently locked in the contract.

IMPACT:

In the event of a discovered vulnerability, the team would be unable to protect remaining assets while preparing a fix. Tokens accidentally sent to the contract (including the main token) cannot be recovered.

RECOMMENDATIONS:

1. Pausable Pattern: Incorporate OpenZeppelin's Pausable contract to allow temporarily halting critical functions:

```
import ''openzeppelin-contracts/contracts/security/Pausable.sol'';

contract VestingManager is Ownable, Pausable {
    // Add pause/unpause functions with appropriate access control
    // Modify claim() to include ''whenNotPaused'' modifier
}
```

2. Emergency Token Recovery: Add a function to recover tokens in emergency situations:

```
function recoverERC20(address tokenAddress, uint256 amount) external
   onlyOwner {
    // Implement appropriate safeguards
    // For managed token, consider preventing full withdrawal of vested
   tokens
    IERC20(tokenAddress).transfer(owner(), amount);
}
```

3.2 Function _createScheduleInternal not checking the beneficiaries is zero address

SEVERITY: MEDIUM **STATUS:** Fixed

PATH:

`src/VestingManager.sol`

DESCRIPTION:

beneficiary is never checked for zero during construction, so a misconfigured deployment can lock an entire schedule forever and cannot be repaired later because updateBeneficiary rejects zero addresses.

```
function _createScheduleInternal(
      address beneficiary,
      uint256 totalAmount,
      Term[] memory terms
  ) internal {
      uint256 sumWeight = 0;
      for (uint i = 0; i < terms.length; ++i) {
          if (terms[i].period == 0) {
              require(terms[i].num == 1, ''num must 1 when period==0'');
          } else {
              require(terms[i].num > 0, ''num zero'');
          }
          sumWeight += terms[i].percentage;
      }
      require(sumWeight == PPM, ''weights must sum to PPM'');

      VestingSchedule storage s = schedules.push();
      s.beneficiary = beneficiary;
      s.totalAmount = totalAmount;
      s.vestedAmount = 0;
      s.termIndex = 0;
      for (uint i = 0; i < terms.length; ++i) {
          s.terms.push(terms[i]);
      }

      emit ScheduleCreated(schedules.length - 1, totalAmount);
```

```
        emit BeneficiaryUpdated(schedules.length - 1, beneficiary);
    }
```

IMPACT:

Potential fund lockup requiring owner intervention.

RECOMMENDATIONS:

Add a require statement to check that the beneficiary is not the zero address.

3.3 Missing TGE Timestamp Validation

SEVERITY: LOW **STATUS:** Fixed

PATH:

`src/VestingManager.sol`

DESCRIPTION:

The contract does not validate that _tgeTimestamp is in the present or future. Setting a past timestamp would make tokens immediately claimable upon deployment. Potentially allow claiming tokens before they were meant to be available.

```
    constructor(address _token, uint256 _tgeTimestamp, address[] memory
   beneficiaries) Ownable(msg.sender) {
        require(_token != address(0), ''token zero'');
        require(beneficiaries.length == 12, ''need 12 addresses'');
        token = IERC20(_token);
        tgeTimestamp = _tgeTimestamp;
        // ...
    }
```

IMPACT:

Undermines the entire vesting schedule design.

RECOMMENDATIONS:

Add a require statement to validate that _tgeTimestamp is greater than or equal to the current block timestamp:

```
constructor(address _token, uint256 _tgeTimestamp, address[] memory
    beneficiaries) Ownable(msg.sender) {
    // ...
+   require(_tgeTimestamp > block.timestamp, ''tge in past'');
    token = IERC20(_token);
    tgeTimestamp = _tgeTimestamp;
    // ...
}
```

3.4 Missing Check in updateBeneficiary Function

SEVERITY: LOW **STATUS:** Fixed

PATH:

src/VestingManager.sol

DESCRIPTION:

The updateBeneficiary function doesn't verify if the new beneficiary address is different from the current one:

```
function updateBeneficiary(uint256 scheduleId, address newBeneficiary)
    external onlyOwner {
     require(scheduleId < schedules.length, ''invalid index'');
     require(newBeneficiary != address(0), ''new beneficiary is zero'');
     VestingSchedule storage s = schedules[scheduleId];
     s.beneficiary = newBeneficiary;  // No check if it's the same address
     emit BeneficiaryUpdated(scheduleId, newBeneficiary);
}
```

IMPACT:

It wastes gas by updating storage even when no change occur. It emits misleading events suggesting a beneficiary change when none happens.

RECOMMENDATIONS:

Add a check to verify the new address is different from the current one:

```
function updateBeneficiary(uint256 scheduleId, address newBeneficiary)
    external onlyOwner {
     require(scheduleId < schedules.length, ''invalid index'');
     require(newBeneficiary != address(0), ''new beneficiary is zero'');
     VestingSchedule storage s = schedules[scheduleId];
     require(s.beneficiary != newBeneficiary, ''same beneficiary'');
     s.beneficiary = newBeneficiary;
     emit BeneficiaryUpdated(scheduleId, newBeneficiary);
 }
```

3.5 Remove no use contract in the project

SEVERITY: INFO **STATUS:** Fixed

PATH:

```
src/Counter.sol
```

DESCRIPTION:

Counter.sol is useless. We should remove it in the project directory.

4. CONCLUSION

In this audit, we thoroughly analyzed **Nubila Token** smart contract implementation. The problems found are described and explained in detail in Section 3. The problems found in the audit have been communicated to the project leader. We therefore consider the audit result to be **PASSED**.

To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

5. APPENDIX

5.1 Basic Coding Assessment

5.1.1 Apply Verification Control

Description	The security of apply verification
Result	Not found
Severity	CRITICAL

5.1.2 Authorization Access Control

Description	Permission checks for external integral functions
Result	Not found
Severity	CRITICAL

5.1.3 Forged Transfer Vulnerability

Description	Assess whether there is a forged transfer notification vulnerability in the contract
Result	Not found
Severity	CRITICAL

5.1.4 Transaction Rollback Attack

Description	Assess whether there is transaction rollback attack vulnerability in the contract
Result	Not found
Severity	CRITICAL

5.1.5 Transaction Block Stuffing Attack

Description	Assess whether there is transaction blocking attack vulnerability
Result	Not found
Severity	CRITICAL

5.1.6 Soft Fail Attack Assessment

Description	Assess whether there is soft fail attack vulnerability
Result	Not found
Severity	CRITICAL

5.1.7 Hard Fail Attack Assessment

Description	Examine for hard fail attack vulnerability
Result	Not found
Severity	CRITICAL

5.1.8 Abnormal Memo Assessment

Description	Assess whether there is abnormal memo vulnerability in the contract
Result	Not found
Severity	CRITICAL

5.1.9 Abnormal Resource Consumption

Description	Examine whether abnormal resource consumption in contract processing
Result	Not found
Severity	CRITICAL

5.1.10 Random Number Security

Description	Examine whether the code uses insecure random number
Result	Not found
Severity	CRITICAL

5.2 Advanced Code Scrutiny

5.2.1 Cryptography Security

Description	Examine for weakness in cryptograph implementation
Result	Not found
Severity	HIGH

5.2.2 Account Permission Control

Description	Examine permission control issue in the contract
Result	Not found
Severity	MEDIUM

5.2.3 Malicious Code Behavior

Description	Examine whether sensitive behavior present in the code
Result	Not found
Severity	MEDIUM

5.2.4 Sensitive Information Disclosure

Description	Examine whether sensitive information disclosure issue present in the code
Result	Not found
Severity	MEDIUM

5.2.5 System API

Description	Examine whether system API application issue present in the code
Result	Not found
Severity	LOW

6. 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 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 without ExVul'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 considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts ExVul 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. ExVul's position is that each company and individual are responsible for their own due diligence and continuous security. ExVul'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.

7. REFERENCES

[1] MITRE. CWE-191: Integer Underflow (Wrap or Wraparound). https://cwe.mitre.org/data/definitions/191.html.

[2] MITRE. CWE-197: Numeric Truncation Error. https://cwe.mitre.org/data/definitions/197.html.

[3] MITRE. CWE-400: Uncontrolled Resource Consumption. https://cwe.mitre.org/data/definitions/400.html.

[4] MITRE. CWE-440: Expected Behavior Violation. https://cwe.mitre.org/data/definitions/440.html.

[5] MITRE. CWE-684: Protection Mechanism Failure. https://cwe.mitre.org/data/definitions/693.html.

[6] MITRE. CWE CATEGORY: 7PK - Security Features. https://cwe.mitre.org/data/definitions/254.html.

[7] MITRE. CWE CATEGORY: Behavioral Problems. https://cwe.mitre.org/data/definitions/438.html.

[8] MITRE. CWE CATEGORY: Numeric Errors. https://cwe.mitre.org/data/definitions/189.html.

[9] MITRE. CWE CATEGORY: Resource Management Errors. https://cwe.mitre.org/data/definitions/399.html.

[10] OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology

8. About Exvul Security

Premier Security for the Web3 Ecosystem

ExVul is a premier Web3 security firm committed to forging a secure and trustworthy decentralized ecosystem. Our elite team consists of security veterans from world-leading technology and blockchain security firms, including Huawei, YBB Captical, Qihoo 360, Amber, ByteDance, MoveBit, and PeckShield. Team member Nolan is ranked as a top-40 whitehat on Immunefi and is the platform's sole All-Star in the APAC region.

Our expertise covers the full spectrum of Web3 security. We conduct **meticulous smart contract audits**, having fortified thousands of projects on chains like Evm, Solana, Aptos, Sui etc. Our **Blockchain Protocol Audits** secure the core infrastructure of L1/L2 by uncovering deep-seated vulnerabilities. We also offer **comprehensive wallet audits** to protect user assets and provide **proactive web3 pentest**, enabling partners to neutralize threats before they strike.

Trusted by industry leaders, ExVul is the security partner for **OKX, Bitget, Cobo, Infini, Stacks, Aptos, Sui, CoreDAO, Sei** etc.

Contact

🌐 Website
www.exvul.com

✉ Email
contact@exvul.com

𝕏 Twitter
@EXVULSEC

Github
github.com/EXVUL-Sec

ExVul