ExVul

SMART CONTRACT AUDIT REPORT

Nubila

OCTOBER 2025

Contents

1. EXECUTIVE SUMMARY

ExVul Web3 Security was engaged by **Nubila** 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	Informational	Low	Medium	High
High	INFO	MEDIUM	HIGH	CRITICAL
Medium	INFO	LOW	MEDIUM	HIGH
Low	INFO	LOW	LOW	MEDIUM

IMPACT

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	11/10/2025 - 14/10/2025	Solidity

Repository
https://github.com/Nubila/nubila-contract

Commit Hash
84cdb25d4d5ce5ca548f61aef27b2ca5946e651e

2.2 Summary

Severity	Found
CRITICAL	0
HIGH	2
MEDIUM	2
LOW	4
INFO	0

2.3 Key Findings

Severity	Findings Title	Status
HIGH	Blacklist Lock Bypass	Fixed
HIGH	redistributeLockedAmount Fails to Access Locked Funds	Fixed
MEDIUM	autoValidate forgeable validators	Fixed
MEDIUM	rescueTokens allows sending tokens to blacklisted addresses	Fixed
LOW	Missing Zero-Address Validation in Admin Transfer	Fixed
LOW	Should make sure the _lockDuration is not set to 0 during initialize	Fixed
LOW	Initial lock duration is not validated	Fixed
LOW	BLACKLIST_MANAGER can DoS the contract	Fixed

Table 2.3: Key Audit Findings

3. DETAILED DESCRIPTION OF FINDINGS

3.1 Blacklist Lock Bypass

SEVERITY: HIGH **STATUS:** Fixed

PATH:

`contracts/StakedNUBI.sol`

DESCRIPTION:

Blacklisted stakers can still exit by calling claim and routing unlocked shares to an unblacklisted address. The claim function does not check the caller's blacklist status, and the _beforeTokenTransfer hook is bypassed because the msg.sender of the internal transfer is the contract itself, not the original blacklisted caller.

The vulnerable code section:

```
// contracts/StakedNUBI.sol
function claim(address receiver) external nonReentrant {
     UserLocks storage userLocks = locks[msg.sender];
     // ...
     IERC20Upgradeable(address(this)).transfer(receiver, shares);
     // ...
}

// contracts/StakedNUBI.sol
function _beforeTokenTransfer(address from, address to, uint256) internal
    virtual override {
     if (hasRole(BLACKLISTED_ROLE, msg.sender)) {
          revert OperationNotAllowed();
     }
     if (hasRole(BLACKLISTED_ROLE, from) && to != address(0)) {
          revert OperationNotAllowed();
     }
     if (hasRole(BLACKLISTED_ROLE, to)) {
          revert OperationNotAllowed();
     }
}
```

IMPACT:

A supposedly frozen user can fully recover their funds after the lock period despite being blacklisted. This defeats the purpose of the blacklist compliance control and makes the redistributeLockedAmount function unusable in practice.

RECOMMENDATIONS:

Add an explicit check to forbid blacklisted callers at the beginning of the claim function:

```
 function claim(address receiver) external nonReentrant {
+    if (hasRole(BLACKLISTED_ROLE, msg.sender)) revert
    OperationNotAllowed();
     UserLocks storage userLocks = locks[msg.sender];
     if (userLocks.tail <= userLocks.head) {
         revert NoSharesLocked();
```

3.2 redistributeLockedAmount Fails to Access Locked Funds

SEVERITY: HIGH **STATUS:** Fixed

PATH:

`contracts/StakedNUBI.sol`

DESCRIPTION:

redistributeLockedAmount incorrectly uses balanceOf(from), which only finds claimed shares in a user's wallet, not the actual locked funds held by the contract in the locks queue.

The vulnerable code section:

```
// contracts/StakedNUBI.sol
function redistributeLockedAmount(address from, address to) external
    nonReentrant onlyRole(DEFAULT_ADMIN_ROLE) {
     if (!hasRole(BLACKLISTED_ROLE, from) || hasRole(BLACKLISTED_ROLE,
    to)) revert OperationNotAllowed();

     uint256 amountToDistribute = balanceOf(from);

     if (amountToDistribute == 0) revert InvalidAmount();
     // ...
}
```

IMPACT:

The function fails to seize locked (unclaimed) funds from blacklisted users, rendering the admin's primary enforcement tool against them inoperable.

RECOMMENDATIONS:

Redesign the function to seize funds from both the locks storage queue and the user's external balance. The function should access the user's locked shares directly from the locks mapping and process them appropriately.

3.3 autoValidate forgeable validators

SEVERITY: MEDIUM **STATUS:** Fixed

PATH:

`contracts/Weather.sol`

DESCRIPTION:

The autoValidate method in the Weather contract allows for third-party automatic validation. An external validator address is passed in, but no permission check is set.

The vulnerable code section:

```
// Weather.sol
function autoValidate(
    address validator,
    string calldata _serialNumber,
    string calldata _dataID,
    bool _isAccurate
) external {
    emit ValidateWeatherData(validator, _serialNumber, _dataID,
   _isAccurate, true);
}
```

IMPACT:

Users can forge validator addresses at will, resulting in falsified Event data.

RECOMMENDATIONS:

Verify msg.sender to ensure it is legal. Add proper access control to ensure only authorized validators can call this function:

```
 function autoValidate(
     address validator,
     string calldata _serialNumber,
     string calldata _dataID,
      bool _isAccurate
  ) external {
 +    require(msg.sender == validator, "Unauthorized validator");
 +    require(hasRole(VALIDATOR_ROLE, msg.sender), "Not a valid
    validator");
      emit ValidateWeatherData(validator, _serialNumber, _dataID,
    _isAccurate, true);
  }
```

3.4 rescueTokens allows sending tokens to blacklisted addresses

SEVERITY: MEDIUM **STATUS:** Fixed

PATH:

`contracts/StakedNUBI.sol`

DESCRIPTION:

The rescueTokens function does not validate the to address against the BLACKLISTED_ROLE. As a result, the admin can transfer arbitrary ERC20 tokens held by the contract to an address that has been blacklisted.

The vulnerable code section:

```
function rescueTokens(
    address token,
    uint256 amount,
    address to
) external nonReentrant onlyRole(DEFAULT_ADMIN_ROLE) {
    if (address(token) == asset()) revert InvalidToken();
    IERC20Upgradeable(token).safeTransfer(to, amount);
    emit TokensRescued(token, to, amount);
}
```

IMPACT:

Blacklisted addresses can receive value from this contract via admin-triggered rescues, contrary to the blacklist intent.

RECOMMENDATIONS:

Add a blacklist guard to rescueTokens to ensure tokens cannot be sent to blacklisted addresses:

```
function rescueTokens(
    address token,
    uint256 amount,
    address to
```

```
 ) external nonReentrant onlyRole(DEFAULT_ADMIN_ROLE) {
     if (address(token) == asset()) revert InvalidToken();
+    require(!hasRole(BLACKLISTED_ROLE, to), "OperationNotAllowed");
     IERC20Upgradeable(token).safeTransfer(to, amount);
     emit TokensRescued(token, to, amount);
 }
```

3.5 Missing Zero-Address Validation in Admin Transfer

SEVERITY: LOW **STATUS:** Fixed

PATH:

`contracts/SingleAdminAccessControlUpgradeable.sol`

DESCRIPTION:

The transferAdmin function does not prevent transferring the admin role to the zero address.

IMPACT:

This could lead to a dead-end in the admin transfer process, requiring the current admin to correct it.

RECOMMENDATIONS:

Add a check to ensure the newAdmin address is not the zero address:

```
 function transferAdmin(address newAdmin) external
   onlyRole(DEFAULT_ADMIN_ROLE) {
+    if (newAdmin == address(0)) revert InvalidAdminChange();
     if (newAdmin == msg.sender) revert InvalidAdminChange();
     _pendingDefaultAdmin = newAdmin;
     emit AdminTransferRequested(_currentDefaultAdmin, newAdmin);
 }
```

3.6 Should make sure the _lockDuration is not set to 0 during initialize

SEVERITY: LOW **STATUS:** Fixed

PATH:

`contracts/StakedNUBI.sol`

DESCRIPTION:

The initializer does not validate the _lockDuration parameter. If initialized with _lockDuration = 0, users can deposit and then immediately claim their funds because the lockEnd calculation becomes:

`userLocks.queue[userLocks.tail].lockEnd = uint64(block.timestamp) + _lockDuration`

When _lockDuration is 0, this means users can claim immediately, bypassing the intended lock period.

The vulnerable code section:

```
function initialize(
    IERC20Upgradeable _asset,
    address _initialRewarder,
    address _owner,
    uint64 _vestingPeriod,
    uint64 _lockDuration
) public initializer {
    __ReentrancyGuard_init();
    __ERC20_init(”Staked NUBI”, ”stNUBI”);
    __ERC20Permit_init(”stNUBI”);
    __ERC4626_init(_asset);

    if (_owner == address(0) || _initialRewarder == address(0) ||
   address(_asset) == address(0)) {
        revert InvalidZeroAddress();
    }
    _updateVestingPeriod(_vestingPeriod);
    _grantRole(REWARDER_ROLE, _initialRewarder);
    _grantRole(DEFAULT_ADMIN_ROLE, _owner);
    lockDuration = _lockDuration;
    emit LockDurationUpdated(0, lockDuration);
}
```

IMPACT:

If the lock duration is set to 0 during initialization, users can bypass the lock mechanism entirely and claim funds immediately after depositing.

RECOMMENDATIONS:

Add validation to ensure _lockDuration is greater than 0 during initialization to enforce the intended lock period mechanism.

3.7 Initial lock duration is not validated

SEVERITY: LOW **STATUS:** Fixed

PATH:

`contracts/StakedNUBI.sol`

DESCRIPTION:

The initialize function does not validate the _lockDuration parameter against MAX_LOCK_DURATION, allowing the deployer to set an arbitrarily long initial lock time.

The vulnerable code section:

```
// contracts/StakedNUBI.sol
function initialize(
    //...
    uint64 _lockDuration
) public initializer {
    //...
    lockDuration = _lockDuration;
    emit LockDurationUpdated(0, lockDuration);
}
```

IMPACT:

A malicious deployment could trap early users' funds for a period longer than the advertised maximum of 180 days.

RECOMMENDATIONS:

Add a check in the initialize function to ensure the initial _lockDuration does not exceed MAX_LOCK_DURATION:

```
     _grantRole(REWARDER_ROLE, _initialRewarder);
     _grantRole(DEFAULT_ADMIN_ROLE, _owner);
+    if (_lockDuration > MAX_LOCK_DURATION) {
+        revert InvalidLockDuration();
+     }
     lockDuration = _lockDuration;
     emit LockDurationUpdated(0, lockDuration);
```

3.8 BLACKLIST_MANAGER can DoS the contract

SEVERITY: LOW **STATUS:** Fixed

PATH:

`contracts/StakedNUBI.sol`

DESCRIPTION:

The addToBlacklist function does not prevent blacklisting the contract itself. If the contract address is blacklisted, the _beforeTokenTransfer hook will block all share transfers to and from the contract, causing deposit() and claim() to fail for all users.

The vulnerable code section:

```
function addToBlacklist(address target) external nonReentrant
   onlyRole(BLACKLIST_MANAGER_ROLE) notOwner(target) {
    // Missing check for `target != address(this)`
    _grantRole(BLACKLISTED_ROLE, target);
}
```

IMPACT:

A malicious BLACKLIST_MANAGER can disable all staking and claiming functions.

RECOMMENDATIONS:

Add a check to addToBlacklist to prevent the contract from blacklisting itself:

```
 function addToBlacklist(address target) external nonReentrant
   onlyRole(BLACKLIST_MANAGER_ROLE) notOwner(target) {
+    if (target == address(this)) revert("Cannot blacklist the contract
   itself");
     _grantRole(BLACKLISTED_ROLE, target);
 }
```

4. CONCLUSION

In this audit, we thoroughly analyzed **Nubila** 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

Twitter
@EXVULSEC

Github
github.com/EXVUL-Sec

ExVul