StreamLink Token

Smart Contract Review

Deliverable: Smart Contract Audit Report

Security Report

September 2021

Disclaimer

The information and views set out in this publication are those of the author(s) and do not necessarily reflect the official opinion of the Company. The content, conclusions and recommendations set out in this publication are elaborated in the specific for only project.

eNebula Solutions does not guarantee the authenticity of the project or organization or team of members that is connected/owner behind the project or nor accuracy of the data included in this study. All representations, warranties, undertakings and guarantees relating to the report are excluded, particularly concerning – but not limited to – the qualities of the assessed projects and products. Neither the Company nor any personating on the Company's behalf may be held responsible for the use that may be made of the information contained herein.

eNebula Solutions retains the right to display audit reports and other content elements as examples of their work in their portfolio and as content features in other projects with protecting all security purpose of customer. The report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities fixed - upon a decision of the Customer.

Report Summary

Title	**StreamLink Token Smart Contract Audit**		
Project Owner	StreamLink		
Type	Public		
Reviewed by	Vatsal Raychura	**Revision date**	22/09/2021
Approved by	eNebula Solutions Private Limited	**Approval date**	22/09/2021
		№ Pages	**28**

Overview

Background

StreamLink requested that eNebula Solutions perform an Extensive Smart Contract audit of their StreamLink Token Smart Contract.

Project Dates

The following is the project schedule for this review and report:

- **September 22**: Smart Contract Review Completed *(Completed)*
- **September 22**: Delivery of Smart Contract Audit Report *(Completed)*

Review Team

The following eNebula Solutions team member participated in this review:

- Sejal Barad, Security Researcher and Engineer
- Vatsal Raychura, Security Researcher and Engineer

Coverage

Target Specification and Revision

For this audit, we performed research, investigation, and review of the smart contract of StreamLink Token.

The following documentation repositories were considered in-scope for the review:

- StreamLink Project: https://bscscan.com/token/0x2b887340e21dc89fa2502b1d8b2722726e3decda

Introduction

Given the opportunity to review StreamLink Project's smart contract source code, we in the report outline our systematic approach to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistencies between smart contract code and design document, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts is ready to launch after resolving the mentioned issues, there are no critical or high issues found related to business logic, security or performance.

About StreamLink: -

Item	Description
Issuer	StreamLink
Website	www.streamlink.app
Type	BEP20
Platform	Solidity
Audit Method	Whitebox
Latest Audit Report	September 22, 2021

The Test Method Information: -

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open-source code, non-open-source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant effect on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project party should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.

The Full List of Check Items:

Category	Check Item
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	MONEY-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead of Transfer
	Costly Loop
	(Unsafe) Use of Untrusted Libraries
	(Unsafe) Use of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses
Semantic Consistency Checks	Semantic Consistency Checks
	Business Logics Review

Advanced DeFi Scrutiny	Functionality Checks
	Authentication Management
	Access Control & Authorization
	Oracle Security
	Digital Asset Escrow
	Kill-Switch Mechanism
	Operation Trails & Event Generation
	ERC20 Idiosyncrasies Handling
	Frontend-Contract Integration
	Deployment Consistency
	Holistic Risk Management
Additional Recommendations	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

Common Weakness Enumeration (CWE) Classifications Used in This Audit:

Category	Summary
Configuration	Weaknesses in this category are typically introduced during the configuration of the software.
Data Processing Issues	Weaknesses in this category are typically found in functionality that processes data.
Numeric Errors	Weaknesses in this category are related to improper calculation or conversion of numbers.
Security Features	Weaknesses in this category are concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management. (Software security is not security software.)
Time and State	Weaknesses in this category are related to the improper management of time and state in an environment that supports simultaneous or near-simultaneous computation by multiple systems, processes, or threads.
Error Conditions, Return Values, Status Codes	Weaknesses in this category include weaknesses that occur if a function does not generate the correct return/status code, or if the application does not handle all possible return/status codes that could be generated by a function.
Resource Management	Weaknesses in this category are related to improper management of system resources.

Behavioral Issues	Weaknesses in this category are related to unexpected behaviors from code that an application uses.
Business Logics	Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application.
Initialization and Cleanup	Weaknesses in this category occur in behaviors that are used for initialization and breakdown.
Arguments and Parameters	Weaknesses in this category are related to improper use arguments or parameters within function calls.
Expression Issues	Weaknesses in this category are related to incorrectly written expressions within code.
Coding Practices	Weaknesses in this category are related to coding practices that are deemed unsafe and increase the chances that an ex pilotable vulnerability will be present in the application. They may not directly introduce a vulnerability, but indicate the product has not been carefully developed or maintained.

Findings

Summary

Here is a summary of our findings after analyzing the StreamLink Token Smart Contract. During the first phase of our audit, we studied the smart contract source code and ran our in-house static code analyzer through the Specific tool. The purpose here is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by tool. We further manually review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

Severity	No. of Issues
Critical	**0**
High	**0**
Medium	**0**
Low	**6**
Total	**6**

We have so far identified that there are potential issues with severity of **0 Critical, 0 High, 0 Medium, and 6 Low**. Overall, these smart contracts are well- designed and engineered, though the implementation can be improved and bug free by common recommendations given under POCs.

Functional Overview

($) = payable function # = non-constant function	[Pub] public [Ext] external [Prv] private [Int] internal

```
+ [Lib] SafeCast
   - [Int] toUint224
   - [Int] toUint128
   - [Int] toUint96
   - [Int] toUint64
   - [Int] toUint32
   - [Int] toUint16
   - [Int] toUint8
   - [Int] toUint256
   - [Int] toInt128
   - [Int] toInt64
   - [Int] toInt32
   - [Int] toInt16
   - [Int] toInt8
   - [Int] toInt256

 + [Lib] ECDSA
   - [Prv] _throwError
   - [Int] tryRecover
   - [Int] recover
   - [Int] tryRecover
   - [Int] recover
```

```
  - [Int] tryRecover
  - [Int] recover
  - [Int] toEthSignedMessageHash
  - [Int] toTypedDataHash

+  EIP712
  - [Pub] <Constructor> #
  - [Int] _domainSeparatorV4
  - [Prv] _buildDomainSeparator
  - [Int] _hashTypedDataV4

+ [Int] IERC20Permit
  - [Ext] permit #
  - [Ext] nonces
  - [Ext] DOMAIN_SEPARATOR

+ [Lib] Counters
  - [Int] current
  - [Int] increment #
  - [Int] decrement #
  - [Int] reset #

+ [Lib] Math
  - [Int] max
  - [Int] min
  - [Int] average
  - [Int] ceilDiv

+ [Lib] Arrays
  - [Int] findUpperBound
```

+ [Int] IERC20
  - [Ext] totalSupply
  - [Ext] balanceOf
  - [Ext] transfer #
  - [Ext] allowance
  - [Ext] approve #
  - [Ext] transferFrom #

+ [Int] IERC20Metadata (IERC20)
  - [Ext] name
  - [Ext] symbol
  - [Ext] decimals

+ Context
  - [Int] _msgSender
  - [Int] _msgData

+ ERC20 (Context, IERC20, IERC20Metadata)
  - [Pub] <Constructor> #
  - [Pub] name
  - [Pub] symbol
  - [Pub] decimals
  - [Pub] totalSupply
  - [Pub] balanceOf
  - [Pub] transfer #
  - [Pub] allowance
  - [Pub] approve #
  - [Pub] transferFrom #
  - [Pub] increaseAllowance #
  - [Pub] decreaseAllowance #
  - [Int] _transfer #

```
  - [Int] _mint #
  - [Int] _burn #
  - [Int] _approve #
  - [Int] _beforeTokenTransfer #
  - [Int] _afterTokenTransfer #

+  ERC20Burnable (Context, ERC20)
  - [Pub] burn #
  - [Pub] burnFrom #

+  ERC20Snapshot (ERC20)
  - [Int] _snapshot #
  - [Int] _getCurrentSnapshotId
  - [Pub] balanceOfAt
  - [Pub] totalSupplyAt
  - [Int] _beforeTokenTransfer #
  - [Prv] _valueAt
  - [Prv] _updateAccountSnapshot #
  - [Prv] _updateTotalSupplySnapshot #
  - [Prv] _updateSnapshot #
  - [Prv] _lastSnapshotId

+  Ownable (Context)
  - [Pub] <Constructor> #
  - [Pub] owner
  - [Pub] renounceOwnership #
    - modifiers: onlyOwner
  - [Pub] transferOwnership #
    - modifiers: onlyOwner
  - [Prv] _setOwner #
```

+ ERC20Permit (ERC20, IERC20Permit, EIP712)
  - [Pub] <Constructor> #
    - modifiers: EIP712
  - [Pub] permit #
  - [Pub] nonces
  - [Ext] DOMAIN_SEPARATOR
  - [Int] _useNonce #

+ ERC20Votes (ERC20Permit)
  - [Pub] checkpoints
  - [Pub] numCheckpoints
  - [Pub] delegates
  - [Pub] getVotes
  - [Pub] getPastVotes
  - [Pub] getPastTotalSupply
  - [Prv] _checkpointsLookup
  - [Pub] delegate #
  - [Pub] delegateBySig #
  - [Int] _maxSupply
  - [Int] _mint #
  - [Int] _burn #
  - [Int] _afterTokenTransfer #
  - [Int] _delegate #
  - [Prv] _moveVotingPower #
  - [Prv] _writeCheckpoint #
  - [Prv] _add
  - [Prv] _subtract

+ StreamLink (ERC20, ERC20Burnable, ERC20Snapshot, Ownable, ERC20Permit, ERC20Votes)
  - [Pub] <Constructor> #
    - modifiers: ERC20,ERC20Permit

- [Pub] snapshot #
  - modifiers: onlyOwner
- [Pub] mint #
  - modifiers: onlyOwner
- [Int] _beforeTokenTransfer #
- [Int] _afterTokenTransfer #
- [Int] _mint #
- [Int] _burn #

Detailed Results

Issues Checking Status

1. A floating pragma is set.

- SWC ID:103
- Severity: Low
- Location: https://bscscan.com/address/0x2b887340e21dc89fa2502b1d8b2722726e3decda#code
- Relationships: CWE-664: Improper Control of a Resource Through its Lifetime
- Description: The current pragma Solidity directive is ""^0.8.2"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.
- POC:

```

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.2;

```

- Remediations: Lock the pragma version and also consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

2. Weak Sources of Randomness from Chain Attributes

- SWC ID:120
- Severity: Low
- Location: https://bscscan.com/address/0x2b887340e21dc89fa2502b1d8b2722726e3decda#code
- Relationships: CWE-330: Use of Insufficiently Random Values
- Description: A control flow decision is made based on The block.number environment variable. The block.number environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.
- POC:

```
    function toUint32(uint256 value) internal pure returns (uint32) {
        require(value <= type(uint32).max, "SafeCast: value doesn't fit in 32 bits");
        return uint32(value);
    }
```

- Remediations:
  - Using commitment scheme, e.g. RANDAO.
  - Using external sources of randomness via oracles, e.g. Oraclize. Note that this approach requires trusting in oracle, thus it may be reasonable to use multiple oracles.
  - Using Bitcoin block hashes, as they are more expensive to mine.

3. Weak Sources of Randomness from Chain Attributes

- SWC ID:120
- Severity: Low
- Location: https://bscscan.com/address/0x2b887340e21dc89fa2502b1d8b2722726e3decda#code
- Relationships: CWE-330: Use of Insufficiently Random Values
- Description: Potential use of "block.number" as source of randomness. The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.
- POC:

```
function getPastVotes(address account, uint256 blockNumber) public view returns (uint256) {
    require(blockNumber < block.number, "ERC20Votes: block not yet mined");
    return _checkpointsLookup(_checkpoints[account], blockNumber);
}
```

- Remediations:
  - Using commitment scheme, e.g. RANDAO.
  - Using external sources of randomness via oracles, e.g. Oraclize. Note that this approach requires trusting in oracle, thus it may be reasonable to use multiple oracles.
  - Using Bitcoin block hashes, as they are more expensive to mine.

4. Weak Sources of Randomness from Chain Attributes

- SWC ID:120
- Severity: Low
- Location: https://bscscan.com/address/0x2b887340e21dc89fa2502b1d8b2722726e3decda#code
- Relationships: CWE-330: Use of Insufficiently Random Values
- Description: Potential use of "block.number" as source of randomness. The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.
- POC:

```
function getPastTotalSupply(uint256 blockNumber) public view returns (uint256) {
    require(blockNumber < block.number, "ERC20Votes: block not yet mined");
    return _checkpointsLookup(_totalSupplyCheckpoints, blockNumber);
}
```

- Remediations:
  - Using commitment scheme, e.g. RANDAO.
  - Using external sources of randomness via oracles, e.g. Oraclize. Note that this approach requires trusting in oracle, thus it may be reasonable to use multiple oracles.
  - Using Bitcoin block hashes, as they are more expensive to mine.

5. Weak Sources of Randomness from Chain Attributes

- SWC ID:120
- Severity: Low
- Location: https://bscscan.com/address/0x2b887340e21dc89fa2502b1d8b2722726e3decda#code
- Relationships: CWE-330: Use of Insufficiently Random Values
- Description: Potential use of "block.number" as source of randomness. The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.
- POC:

```
        if (pos > 0 && ckpts[pos - 1].fromBlock == block.number) {
            ckpts[pos - 1].votes = SafeCast.toUint224(newWeight);
        } else {
            ckpts.push(Checkpoint({fromBlock: SafeCast.toUint32(block.number), votes: SafeCast.toUint224(newWeight)}));
        }
    }
```

- Remediations:
  - Using commitment scheme, e.g. RANDAO.
  - Using external sources of randomness via oracles, e.g. Oraclize. Note that this approach requires trusting in oracle, thus it may be reasonable to use multiple oracles.
  - Using Bitcoin block hashes, as they are more expensive to mine.

6. Weak Sources of Randomness from Chain Attributes

- SWC ID:120
- Severity: Low
- Location: https://bscscan.com/address/0x2b887340e21dc89fa2502b1d8b2722726e3decda#code
- Relationships: CWE-330: Use of Insufficiently Random Values
- Description: Potential use of "block.number" as source of randomness. The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.
- POC:

```
        if (pos > 0 && ckpts[pos - 1].fromBlock == block.number) {
            ckpts[pos - 1].votes = SafeCast.toUint224(newWeight);
        } else {
            ckpts.push(Checkpoint({fromBlock: SafeCast.toUint32(block.number), votes: SafeCast.toUint224(newWeight)}));
        }
    }
```

- Remediations:
  - Using commitment scheme, e.g. RANDAO.
  - Using external sources of randomness via oracles, e.g. Oraclize. Note that this approach requires trusting in oracle, thus it may be reasonable to use multiple oracles.
  - Using Bitcoin block hashes, as they are more expensive to mine.

Automated Tools Results

Slither: -

```
ERC20Votes._writeCheckpoint(ERC20Votes.Checkpoint[],function(uint256,uint256) returns(uint256),uint256) (StreamLink.sol#1624-1638) uses a dan
gerous strict equality:
        - pos > 0 && ckpts[pos - 1].fromBlock == block.number (StreamLink.sol#1633)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

ERC20Votes._moveVotingPower(address,address,uint256).newWeight_scope_1 (StreamLink.sol#1618) is a local variable never initialized
ERC20Votes._moveVotingPower(address,address,uint256).oldWeight_scope_0 (StreamLink.sol#1618) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
```

```
ERC20Permit.constructor(string).name (StreamLink.sol#1363) shadows:
        - ERC20.name() (StreamLink.sol#812-814) (function)
        - IERC20Metadata.name() (StreamLink.sol#760) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Variable 'ECDSA.tryRecover(bytes32,bytes).r (StreamLink.sol#280)' in ECDSA.tryRecover(bytes32,bytes) (StreamLink.sol#275-304) potentially use
d before declaration: r = mload(uint256)(signature + 0x20) (StreamLink.sol#297)
Variable 'ERC20Votes._moveVotingPower(address,address,uint256).oldWeight (StreamLink.sol#1613)' in ERC20Votes._moveVotingPower(address,addres
s,uint256) (StreamLink.sol#1606-1622) potentially used before declaration: (oldWeight,newWeight) = _writeCheckpoint(_checkpoints[dst],_add,am
ount) (StreamLink.sol#1618)
Variable 'ERC20Votes._moveVotingPower(address,address,uint256).newWeight (StreamLink.sol#1613)' in ERC20Votes._moveVotingPower(address,addres
s,uint256) (StreamLink.sol#1606-1622) potentially used before declaration: (oldWeight,newWeight) = _writeCheckpoint(_checkpoints[dst],_add,am
ount) (StreamLink.sol#1618)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables

ERC20Permit.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (StreamLink.sol#1368-1387) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= deadline,ERC20Permit: expired deadline) (StreamLink.sol#1377)
ERC20Votes.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (StreamLink.sol#1531-1548) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= expiry,ERC20Votes: signature expired) (StreamLink.sol#1539)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

```
Context._msgData() (StreamLink.sol#779-781) is never used and should be removed
Counters.decrement(Counters.Counter) (StreamLink.sol#589-595) is never used and should be removed
Counters.reset(Counters.Counter) (StreamLink.sol#597-599) is never used and should be removed
ECDSA.recover(bytes32,bytes) (StreamLink.sol#320-324) is never used and should be removed
ECDSA.recover(bytes32,bytes32,bytes32) (StreamLink.sol#352-360) is never used and should be removed
ECDSA.toEthSignedMessageHash(bytes32) (StreamLink.sol#422-426) is never used and should be removed
ECDSA.tryRecover(bytes32,bytes) (StreamLink.sol#275-304) is never used and should be removed
ECDSA.tryRecover(bytes32,bytes32,bytes32) (StreamLink.sol#333-345) is never used and should be removed
ERC20Votes._add(uint256,uint256) (StreamLink.sol#1640-1642) is never used and should be removed
ERC20Votes._subtract(uint256,uint256) (StreamLink.sol#1644-1646) is never used and should be removed
Math.ceilDiv(uint256,uint256) (StreamLink.sol#633-636) is never used and should be removed
Math.max(uint256,uint256) (StreamLink.sol#607-609) is never used and should be removed
Math.min(uint256,uint256) (StreamLink.sol#614-616) is never used and should be removed
SafeCast.toInt128(int256) (StreamLink.sol#139-142) is never used and should be removed
SafeCast.toInt16(int256) (StreamLink.sol#193-196) is never used and should be removed
SafeCast.toInt256(uint256) (StreamLink.sol#223-227) is never used and should be removed
SafeCast.toInt32(int256) (StreamLink.sol#175-178) is never used and should be removed
SafeCast.toInt64(int256) (StreamLink.sol#157-160) is never used and should be removed
SafeCast.toInt8(int256) (StreamLink.sol#211-214) is never used and should be removed
SafeCast.toUint128(uint256) (StreamLink.sol#34-37) is never used and should be removed
SafeCast.toUint16(uint256) (StreamLink.sol#94-97) is never used and should be removed
SafeCast.toUint256(int256) (StreamLink.sol#121-124) is never used and should be removed
SafeCast.toUint64(uint256) (StreamLink.sol#64-67) is never used and should be removed
SafeCast.toUint8(uint256) (StreamLink.sol#109-112) is never used and should be removed
SafeCast.toUint96(uint256) (StreamLink.sol#49-52) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.2 (StreamLink.sol#6) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.2 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

```
Variable EIP712._CACHED_DOMAIN_SEPARATOR (StreamLink.sol#448) is not in mixedCase
Variable EIP712._CACHED_CHAIN_ID (StreamLink.sol#449) is not in mixedCase
Variable EIP712._HASHED_NAME (StreamLink.sol#451) is not in mixedCase
Variable EIP712._HASHED_VERSION (StreamLink.sol#452) is not in mixedCase
Variable EIP712._TYPE_HASH (StreamLink.sol#453) is not in mixedCase
Function IERC20Permit.DOMAIN_SEPARATOR() (StreamLink.sol#567) is not in mixedCase
Function ERC20Permit.DOMAIN_SEPARATOR() (StreamLink.sol#1400-1402) is not in mixedCase
Variable ERC20Permit._PERMIT_TYPEHASH (StreamLink.sol#1355-1356) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

StreamLink.constructor() (StreamLink.sol#1650-1652) uses literals with too many digits:
        - _mint(msg.sender,110000000 * 10 ** decimals()) (StreamLink.sol#1651)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

ERC20Permit._PERMIT_TYPEHASH (StreamLink.sol#1355-1356) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
```

Mythril: -

```
SWC ID: 120
Severity: Low
Contract: StreamLink
Function name: constructor
PC address: 1717
Estimated Gas Usage: 44459 - 158492
A control flow decision is made based on The block.number environment variable.
The block.number environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit
, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of ear
lier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a cert
ain level of trust into miners.
--------------------
In file: StreamLink.sol:80

require(value <= type(uint32).max, "SafeCast: value doesn't fit in 32 bits")

--------------------
Initial State:

Account: [CREATOR], balance: 0x1, nonce:0, storage:{}
Account: [ATTACKER], balance: 0x0, nonce:0, storage:{}
Account: [SOMEGUY], balance: 0x0, nonce:0, storage:{}

Transaction Sequence:

Caller: [CREATOR], calldata: , value: 0x0
```

Solhint: -

Linter results:

```
StreamLink.sol:584:18: Error: Parse error: missing ';' at '{'

StreamLink.sol:592:18: Error: Parse error: missing ';' at '{'

StreamLink.sol:909:18: Error: Parse error: missing ';' at '{'

StreamLink.sol:950:18: Error: Parse error: missing ';' at '{'

StreamLink.sol:983:18: Error: Parse error: missing ';' at '{'

StreamLink.sol:1032:18: Error: Parse error: missing ';' at '{'

StreamLink.sol:1132:18: Error: Parse error: missing ';' at '{'
```

Basic Coding Bugs

1. **Constructor Mismatch**
    - Description: Whether the contract name and its constructor are not identical to each other.
    - Result: PASSED
    - Severity: Critical

2. **Ownership Takeover**
    - Description: Whether the set owner function is not protected.
    - Result: PASSED
    - Severity: Critical

3. **Redundant Fallback Function**
    - Description: Whether the contract has a redundant fallback function.
    - Result: PASSED
    - Severity: Critical

4. **Overflows & Underflows**
    - Description: Whether the contract has general overflow or underflow vulnerabilities
    - Result: PASSED
    - Severity: Critical

5. **Reentrancy**
    - Description: Reentrancy is an issue when code can call back into your contract and change state, such as withdrawing ETHs.
    - Result: PASSED
    - Severity: Critical

6. **MONEY-Giving Bug**
    - Description: Whether the contract returns funds to an arbitrary address.
    - Result: PASSED
    - Severity: High

7. **Blackhole**
    - Description: Whether the contract locks ETH indefinitely: merely in without out.
    - Result: PASSED
    - Severity: High

8. **Unauthorized Self-Destruct**
    - Description: Whether the contract can be killed by any arbitrary address.
    - Result: PASSED
    - Severity: Medium

9. **Revert DoS**
    - Description: Whether the contract is vulnerable to DoS attack because of unexpected revert.
    - Result: PASSED
    - Severity: Medium

10. **Unchecked External Call**
    - Description: Whether the contract has any external call without checking the return value.
    - Result: PASSED
    - Severity: Medium

11. **Gasless Send**
    - Description: Whether the contract is vulnerable to gasless send.
    - Result: PASSED
    - Severity: Medium

12. **Send Instead of Transfer**
    - Description: Whether the contract uses send instead of transfer.
    - Result: PASSED
    - Severity: Medium

13. Costly Loop

- Description: Whether the contract has any costly loop which may lead to Out-Of-Gas exception.
- Result: PASSED
- Severity: Medium

14. (Unsafe) Use of Untrusted Libraries

- Description: Whether the contract use any suspicious libraries.
- Result: PASSED
- Severity: Medium

15. (Unsafe) Use of Predictable Variables

- Description: Whether the contract contains any randomness variable, but its value can be predicated.
- Result: PASSED
- Severity: Medium

16. Transaction Ordering Dependence

- Description: Whether the final state of the contract depends on the order of the transactions.
- Result: PASSED
- Severity: Medium

17. Deprecated Uses

- Description: Whether the contract use the deprecated tx.origin to perform the authorization.
- Result: PASSED
- Severity: Medium

Semantic Consistency Checks

- Description: Whether the semantic of the white paper is different from the implementation of the contract.
- Result: PASSED
- Severity: Critical

Conclusion

In this audit, we thoroughly analyzed StreamLink Token Smart Contract. The current code base is well organized but there are promptly some High and low-level issues found in the first phase of Smart Contract Audit.

Meanwhile, we need to emphasize that smart contracts as a whole are still in an early, but exciting stage of development. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

About eNebula Solutions

We believe that people have a fundamental need to security and that the use of secure solutions enables every person to more freely use the Internet and every other connected technology. We aim to provide security consulting service to help others make their solutions more resistant to unauthorized access to data & inadvertent manipulation of the system. We support teams from the design phase through the production to launch and surely after.

The eNebula Solutions team has skills for reviewing code in C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, and JavaScript for common security vulnerabilities & specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture, including in cryptocurrency, blockchains, payments, and smart contracts. Additionally, the team can utilize various tools to scan code & networks and build custom tools as necessary.

Although we are a small team, we surely believe that we can have a momentous impact on the world by being translucent and open about the work we do.

For more information about our security consulting,
please mail us at – contact@enebula.in