BEOSIN
Blockchain Security

StreamNFT

Smart Contract Security Audit

No. 202311171648

Nov 17th, 2023

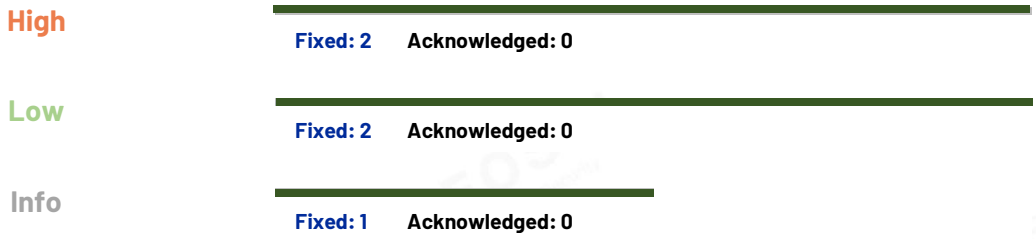
SECURING BLOCKCHAIN ECOSYSTEM

WWW.BEOSIN.COM

Contents

Summary of Audit Results

After auditing, 2 High-risk, 2 Low-risk and 1 Info-risk items were identified in the StreamNFT project. Specific audit details will be presented in the Findings section. Users should pay attention to the following aspects when interacting with this project:



- **Project Description:**

Business overview

StreamNFT is a utility protocol that encompasses P2P NFT rental and P2P Loans. The project operates like a proxy, using the main contract Diamond to make delegate calls to other implementation contracts. Both data and tokens are saved by Diamond.

Provider users can deposit ETH into the contract for lending, and users holding NFT (lender) can stake NFT into the contract and lend and rent. Lender can obtain the specified amount of ETH from the contract when borrowing, and the NFT status is updated to LOAN. Before the loan settlement time, the ETH with interest is paid to the provider through the `repayLoan` function and the LOAN status is deleted. If the NFT is both unborrowed and unleased, the status is INIT, the NFT will be returned to the lender. If the loan settlement time exceeds, the provider can obtain the NFT staked by the lender through the `expireLoan` function.

The lender can freely set the type and quantity of rental tokens and the rental end time when renting. The NFT status is updated to STALE, and the rentee user performs the rental and pays the fee (NFT in the loan state can also be rented, but the rental end time is required to be less than the loan end time). Generally speaking, rentee users cannot directly obtain the NFT staked by the lender. Depending on the status of the domint and the type of NFT, the rentee user may obtain Newly minted ERC7066 tokens and NFT status updates to RENT. The `expireRent` function can be used to update the status of the NFT that has expired at the rental end time to STALE and return the ERC7066 tokens received by the rentee. Lender users can delete the STALE state through the `cancelLendToken` function. If the NFT is in the INIT state, the NFT is returned to the lender.

1 Overview

1.1 Project Overview

Project Name	StreamNFT
Project language	Solidity
Platform	Ethereum, Hedera
Github Link (diamond)	https://github.com/streamnft-tech/EVM/tree/diamond
Commit Hash	6929cf090d63f50e092831b03da85436c2d29c5e c5d54c458b4dd31683edd8258e5aac8154981025
Github Link (diamond-hedera)	https://github.com/streamnft-tech/EVM/tree/diamond-hedera/contracts/facets
Commit Hash	a8227161c129055c4dda1ce33f78f9112851bd98 9bd33a32163642e47f01856ca2b160a68075ae7a af57bc72202ec0223b36eb53cf64d729e1e91e41

1.2 Audit Overview

Audit work duration: Nov 13, 2023 – Nov 17, 2023

Audit team: Beosin Security Team

1.3 Audit Method

The audit methods are as follows:

1. Formal Verification

Formal verification is a technique that uses property-based approaches for testing and verification. Property specifications define a set of rules using Beosin's library of security expert rules. These rules call into the contracts under analysis and make various assertions about their behavior. The rules of the specification play a crucial role in the analysis. If the rule is violated, a concrete test case is provided to demonstrate the violation.

2. Manual Review

Using manual auditing methods, the code is read line by line to identify potential security issues. This ensures that the contract's execution logic aligns with the client's specifications and intentions, thereby safeguarding the accuracy of the contract's business logic.

The manual audit is divided into three groups to cover the entire auditing process:

The Basic Testing Group is primarily responsible for interpreting the project's code and conducting comprehensive functional testing.

The Simulated Attack Group is responsible for analyzing the audited project based on the collected historical audit vulnerability database and security incident attack models. They identify potential attack vectors and collaborate with the Basic Testing Group to conduct simulated attack tests.

The Expert Analysis Group is responsible for analyzing the overall project design, interactions with third parties, and security risks in the on-chain operational environment. They also conduct a review of the entire audit findings.

3. Static Analysis

Static analysis is a method of examining code during compilation or static analysis to detect issues. Beosin-VaaS can detect more than 100 common smart contract vulnerabilities through static analysis, such as reentrancy and block parameter dependency. It allows early and efficient discovery of problems to improve code quality and security.

2 Findings

The code in both branches runs with the same logic, but the code implementation is slightly different due to platform differences, and the following finding is based on the diamond-hedera branch.

Index	Risk description	Severity level	Status
StreamNFT-01	Funds were arbitrarily withdrawn	High	Fixed
StreamNFT-02	Key function missing permission checks	High	Fixed
StreamNFT-03	Re-entry risk	Low	Fixed
StreamNFT-04	Reward sent to address 0	Low	Fixed
StreamNFT-05	Redundant code	Info	Fixed

Finding Details:

[StreamNFT-01] Funds were arbitrarily withdrawn

Severity Level	High
Type	Business Security
Lines	StreamNFT.sol #L212-226
Description	The `shareReward` function in the RentUtils contract can arbitrarily extract all the reward money in the contract. rewardToken and amount are entered by the caller, and all rewardTokens in the contract can be extracted by entering a very large amount of amount. This allows a malicious user to take out all ERC20 tokens and ETH from the contract via the `shareReward` function after using the `lendToken` function.

```
    function shareReward(address tokenAddress, uint256 tokenId,
address rewardToken, uint256 amount) external payable {

        uint256 ownerShare =
amount.mul(StreamStorage.getMapping().assetManager[tokenAddress][to
kenId].rentState.ownerShare).div(100);
        uint256 renteeShare =
amount.mul(100-StreamStorage.getMapping().assetManager[tokenAddress
][tokenId].rentState.ownerShare).div(100);
        StreamLibrary.AssetManager memory _assetManager =
StreamStorage.getMapping().assetManager[tokenAddress][tokenId];

        if(rewardToken != address(0)){
            // Assuming you have an ERC20 interface for the paymentToken
            IERC20(rewardToken).transfer( _assetManager.rentState.re
ntee, renteeShare);
            IERC20(rewardToken).transfer( _assetManager.initializer,
ownerShare);
        } else {
            // Direct ETH transfer
            payable(_assetManager.rentState.rentee).transfer(rentee
Share);
            payable(_assetManager.initializer).transfer(ownerShare)
;
```

```
        }
    }
```

Recommendation It is recommended to modify the logic of the `shareReward` function so that the user can only select the type of reward, not the quantity.

Status **Fixed.** The project states that this function is a design error and that the user should share the reward instead of receiving it from the contract.

```
    function shareReward(address tokenAddress, uint256 tokenId,
address rewardToken, uint256 amount) external payable {
        if(StreamStorage.getMapping().assetManager[tokenAddress][to
kenId].state==StreamLibrary.State.INIT){
            revert ("Invalid State");
        }
        uint256 ownerShare =
amount.mul(StreamStorage.getMapping().assetManager[tokenAddress][to
kenId].rentState.ownerShare).div(100);
        uint256 renteeShare =
amount.mul(100-StreamStorage.getMapping().assetManager[tokenAddress
][tokenId].rentState.ownerShare).div(100);
        StreamLibrary.AssetManager memory _assetManager =
StreamStorage.getMapping().assetManager[tokenAddress][tokenId];
        if(rewardToken != address(0)){
            // Assuming you have an ERC20 interface for the paymentToken
and token approval
            IERC20(rewardToken).transferFrom( msg.sender,_assetManag
er.rentState.rentee, renteeShare);
            IERC20(rewardToken).transferFrom( msg.sender,_assetManag
er.initializer, ownerShare);
        } else {
            // Direct ETH transfer
            StreamLibrary.checkErrorInsufficientFunds(amount);
            payable(_assetManager.rentState.rentee).transfer(rentee
Share);
            payable(_assetManager.initializer).transfer(ownerShare)
;
        }
    }
```

[StreamNFT-02] Key function missing permission checks

Severity Level	High
Type	Business Security
Lines	Stream.sol #L16-28 LoanUtil.sol #L13-16
Description	The `setupConfig` function in the Stream contract has no permission checks and can be called by anyone. And the `checkAdmin` modifier in Stream and LoanUtil is annotated and can't play the role of authentication.

```
    function setupConfig(uint256 rentalFee, address streamNFT, address treasury, address admin, address streamCollection) external{
        StreamStorage.StreamConfig storage config = StreamStorage.getConfig();
        config.streamNFT=streamNFT;
        config.streamRentalFee= rentalFee;
        config.streamTreasury = treasury;
        config.admin=admin;
        config.streamCollection=streamCollection;
    }

    modifier checkAdmin(){
        // if(msg.sender!=admin) revert StreamLibrary.RequiredAdmin();
        _;
    }
```

Recommendation	It is recommended to use a multi-signature wallet to manage the owner permission of this contract.
Status	**Fixed.** Added permission checks.

```
    function setupConfig(uint256 rentalFee, address streamNFT, address treasury, address admin, address streamCollection) external{
        require(msg.sender == LibDiamond.diamondStorage().contractOwner, "LibDiamond: Must be contract owner");
        StreamStorage.StreamConfig storage config = StreamStorage.getConfig();
        config.streamNFT=streamNFT;
```

```
        config.streamRentalFee= rentalFee;
        config.streamTreasury = treasury;
        config.admin=admin;
        config.streamCollection=streamCollection;
    }

    modifier checkAdmin(){
        if(msg.sender!=StreamStorage.getConfig().admin) revert
StreamLibrary.RequiredAdmin();
        _;
    }
```

[StreamNFT-03] Re-entry risk

Severity Level	Low
Type	Business Security
Lines	LoanUtil.sol #L93-128
Description	The `updateOfferCount` and `updateOfferAmount` function in the LoanUtil contract, the transfer occurs before the state variable is changed, although the transfer function only has a gas limit of 2300, there is still a low probability of re-entry risk, according to the code security specification, it is recommended to modify.

```
    function updateOfferAmount(uint256 poolIndex, uint256 offerIndex, uint256 updatedOffer) external payable{
        StreamLibrary.LoanOffer storage offer = StreamStorage.getMapping().loanOfferList[poolIndex-1].loanOffers[offerIndex-1];
        if(offer.bidderPubkey != msg.sender){
            revert StreamLibrary.InvalidUser();
        }
        emit StreamLibrary.UpdateOfferAmount(poolIndex-1, offerIndex-1, updatedOffer);
        if(offer.bidAmount>updatedOffer){
            payable(offer.bidderPubkey).transfer(offer.totalBids*(offer.bidAmount-updatedOffer));
        } else{
            StreamLibrary.checkErrorInsufficientFunds(offer.totalBids*(updatedOffer-offer.bidAmount));
        }
        offer.bidAmount=updatedOffer;
    }
```

Recommendation	It is recommended to use a temporary variable to store offer.totalBids to participate in the transfer calculation and modify offer.totalBids before transfer.
Status	**Fixed.** The temporary variable bidAmount is used and the data is updated before the transfer.

```
    function updateOfferAmount(uint256 poolIndex, uint256 offerIndex, uint256 updatedOffer) external payable{
        StreamLibrary.LoanOffer storage offer =
```

```
StreamStorage.getMapping().loanOfferList[poolIndex].loanOffers[offe
rIndex];
        if(offer.bidderPubkey != msg.sender){
            revert StreamLibrary.InvalidUser();
        }
        emit StreamLibrary.UpdateOfferAmount(poolIndex, offerIndex,
updatedOffer);
        uint256 bidAmount=offer.bidAmount;
        offer.bidAmount=updatedOffer;
        if(bidAmount>updatedOffer){
            payable(offer.bidderPubkey).transfer(offer.totalBids*(b
idAmount-updatedOffer));
        } else{
            StreamLibrary.checkErrorInsufficientFunds(offer.totalBi
ds*(updatedOffer-bidAmount));
        }
    }
```

[StreamNFT-04] Reward sent to address 0

Severity Level	Low
Type	Business Security
Lines	RentUtil.sol #L159-180
Description	The rentee will be set to 0 in the `expireRent` function, when the state is STALE. At this point the `shareReward` function will send the renteeShare to address 0.

```
    function expireRent(address tokenAddress, uint tokenId) external
{
        StreamLibrary.AssetManager memory _assetManager =
StreamStorage.getMapping().assetManager[tokenAddress][tokenId];

        if(_assetManager.rentState.rentExpiry>block.timestamp)
            revert StreamLibrary.PendingExpiry();
        //
require(_assetManager.rentState.rentExpiry<block.timestamp,"R4");
        //
StreamLibrary.checkAssetState(_assetManager.state,[StreamLibrary.St
ate.RENT,StreamLibrary.State.RENT_AND_LOAN]);
        if(_assetManager.state!=StreamLibrary.State.RENT &&
_assetManager.state!=StreamLibrary.State.RENT_AND_LOAN)
            revert StreamLibrary.InvalidAssetState();
        // require(_assetManager.state==StreamLibrary.State.RENT ||
_assetManager.state==StreamLibrary.State.RENT_AND_LOAN, "R7");
        if(_assetManager.state==StreamLibrary.State.RENT){ _assetMan
ager.state=StreamLibrary.State.STALE; }
        else{_assetManager.state=StreamLibrary.State.STALE_AND_LOAN
;}

        emit StreamLibrary.ExpireRent(tokenAddress, tokenId,
_assetManager.rentState.rentee);
        address _rentee = _assetManager.rentState.rentee;
        _assetManager.rentState.rentee=address(0);
        //update storage
        StreamStorage.getMapping().assetManager[tokenAddress][token
Id] = _assetManager;
        // transfer wrapped token if minted
        if(_assetManager.rentState.doMint){
```

```
            StreamLibrary.transferToken(_rentee,address(this),token
Address,tokenId,true,false);
        }
    }
```

Recommendation It is recommended to reset rentee to NFT owner in `expireRent`.

Status **Fixed.**

```
    function expireRent(address tokenAddress, uint tokenId) external
{
        StreamLibrary.AssetManager memory _assetManager =
StreamStorage.getMapping().assetManager[tokenAddress][tokenId];

        if(_assetManager.rentState.rentExpiry>block.timestamp)
            revert ("Expiry pending");
        //
require(_assetManager.rentState.rentExpiry<block.timestamp,"R4");
        //
StreamLibrary.checkAssetState(_assetManager.state,[StreamLibrary.St
ate.RENT,StreamLibrary.State.RENT_AND_LOAN]);
        if(_assetManager.state!=StreamLibrary.State.RENT &&
_assetManager.state!=StreamLibrary.State.RENT_AND_LOAN)
            revert ("Invalid Asset State");
        // require(_assetManager.state==StreamLibrary.State.RENT ||
_assetManager.state==StreamLibrary.State.RENT_AND_LOAN, "R7");
        if(_assetManager.state==StreamLibrary.State.RENT){ _assetMan
ager.state=StreamLibrary.State.STALE; }
        else{_assetManager.state=StreamLibrary.State.STALE_AND_LOAN
;}

        emit StreamLibrary.ExpireRent(tokenAddress, tokenId,
_assetManager.rentState.rentee);
        address _rentee = _assetManager.rentState.rentee;
        _assetManager.rentState.rentee=_assetManager.initializer;
        //update storage
        StreamStorage.getMapping().assetManager[tokenAddress][token
Id] = _assetManager;
        // transfer wrapped token if minted
        if(_assetManager.rentState.doMint){
            StreamLibrary.transferToken(_rentee,address(this),token
```

```
Address,tokenId,true,false);
        }
    }
```

[StreamNFT-05] Redundant code

Severity Level	Info
Type	Coding Conventions
Lines	Stream.sol #L7-8

Description	The StreamLibrary library is referenced twice in Steam.sol.
	`import "../libraries/StreamLibrary.sol";`
	`import "../libraries/StreamLibrary.sol";`
	isFixed and fixedMinutes are not used elsewhere and are redundant.
	`_assetManager.rentState.isFixed=isFixed;`

Recommendation	It is recommended that this be deleted.
Status	**Fixed.** Removed redundant libraries, added isFixed related check in `processRent`.

```
    function processRent(address tokenAddress, uint256 tokenId,
uint256 durationMinutes,bytes32[] calldata proof) external payable
nonReentrant{
        StreamLibrary.AssetManager memory _assetManager =
StreamStorage.getMapping().assetManager[tokenAddress][tokenId];
        uint rent= _assetManager.rentState.rate*durationMinutes;
        uint protocolFee=
rent*StreamStorage.getConfig().streamRentalFee/100;

        if(_assetManager.rentState.validityExpiry<block.timestamp+d
urationMinutes*60)
            revert ("Exceeded validity");

        if(_assetManager.rentState.isFixed &&
_assetManager.rentState.fixedMinutes!=durationMinutes)
            revert StreamLibrary.InvalidTimeDuration();
```

3 Appendix

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts

3.1.1 Metrics

In order to objectively assess the severity level of vulnerabilities in blockchain systems, this report provides detailed assessment metrics for security vulnerabilities in smart contracts with reference to CVSS 3.1(Common Vulnerability Scoring System Ver 3.1).

According to the severity level of vulnerability, the vulnerabilities are classified into four levels: "critical", "high", "medium" and "low". It mainly relies on the degree of impact and likelihood of exploitation of the vulnerability, supplemented by other comprehensive factors to determine of the severity level.

Likelihood \ Impact	Severe	High	Medium	Low
Probable	Critical	High	Medium	Low
Possible	High	Medium	Medium	Low
Unlikely	Medium	Medium	Low	Info
Rare	Low	Low	Info	Info

3.1.2 Degree of impact

- **Severe**

Severe impact generally refers to the vulnerability can have a serious impact on the confidentiality, integrity, availability of smart contracts or their economic model, which can cause substantial economic losses to the contract business system, large-scale data disruption, loss of authority management, failure of key functions, loss of credibility, or indirectly affect the operation of other smart contracts associated with it and cause substantial losses, as well as other severe and mostly irreversible harm.

- **High**

High impact generally refers to the vulnerability can have a relatively serious impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a greater economic loss, local functional unavailability, loss of credibility and other impact to the contract business system.

- **Medium**

Medium impact generally refers to the vulnerability can have a relatively minor impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a small amount of economic loss to the contract business system, individual business unavailability and other impact.

- **Low**

Low impact generally refers to the vulnerability can have a minor impact on the smart contract, which can pose certain security threat to the contract business system and needs to be improved.

3.1.4 Likelihood of Exploitation

- **Probable**

Probable likelihood generally means that the cost required to exploit the vulnerability is low, with no special exploitation threshold, and the vulnerability can be triggered consistently.

- **Possible**

Possible likelihood generally means that exploiting such vulnerability requires a certain cost, or there are certain conditions for exploitation, and the vulnerability is not easily and consistently triggered.

- **Unlikely**

Unlikely likelihood generally means that the vulnerability requires a high cost, or the exploitation conditions are very demanding and the vulnerability is highly difficult to trigger.

- **Rare**

Rare likelihood generally means that the vulnerability requires an extremely high cost or the conditions for exploitation are extremely difficult to achieve.

3.1.5 Fix Results Status

Status	Description
Fixed	The project party fully fixes a vulnerability.
Partially Fixed	The project party did not fully fix the issue, but only mitigated the issue.
Acknowledged	The project party confirms and chooses to ignore the issue.

3.2 Audit Categories

No.	Categories	Subitems
1	Coding Conventions	Compiler Version Security
		Deprecated Items
		Redundant Code
		require/assert Usage
		Gas Consumption
2	General Vulnerability	Integer Overflow/Underflow
		Reentrancy
		Pseudo-random Number Generator (PRNG)
		Transaction-Ordering Dependence
		DoS (Denial of Service)
		Function Call Permissions
		call/delegatecall Security
		Returned Value Security
		s.ContractRef.MsgSender Usage
		Replay Attack
		Overriding Variables
		Third-party Protocol Interface Consistency
3	Business Security	Business Logics
		Business Implementations
		Manipulable Token Price
		Centralized Asset Control
		Asset Tradability
		Arbitrage Attack

Beosin classified the security issues of smart contracts into three categories: Coding Conventions, General Vulnerability, Business Security. Their specific definitions are as follows:

- **Coding Conventions**

Audit whether smart contracts follow recommended language security coding practices. For example, smart contracts developed in Solidity language should fix the compiler version and do not use deprecated keywords.

- **General Vulnerability**

General Vulnerability include some common vulnerabilities that may appear in smart contract projects. These vulnerabilities are mainly related to the characteristics of the smart contract itself, such as integer overflow/underflow and denial of service attacks.

- **Business Security**

Business security is mainly related to some issues related to the business realized by each project, and has a relatively strong pertinence. For example, whether the lock-up plan in the code match the white paper, or the flash loan attack caused by the incorrect setting of the price acquisition oracle.

*Note that the project may suffer stake losses due to the integrated third-party protocol. This is not something Beosin can control. Business security requires the participation of the project party. The project party and users need to stay vigilant at all times.

3.3 Disclaimer

The Audit Report issued by Beosin is related to the services agreed in the relevant service agreement. The Project Party or the Served Party (hereinafter referred to as the "Served Party") can only be used within the conditions and scope agreed in the service agreement. Other third parties shall not transmit, disclose, quote, rely on or tamper with the Audit Report issued for any purpose.

The Audit Report issued by Beosin is made solely for the code, and any description, expression or wording contained therein shall not be interpreted as affirmation or confirmation of the project, nor shall any warranty or guarantee be given as to the absolute flawlessness of the code analyzed, the code team, the business model or legal compliance.

The Audit Report issued by Beosin is only based on the code provided by the Served Party and the technology currently available to Beosin. However, due to the technical limitations of any organization, and in the event that the code provided by the Served Party is missing information, tampered with, deleted, hidden or subsequently altered, the audit report may still fail to fully enumerate all the risks.

The Audit Report issued by Beosin in no way provides investment advice on any project, nor should it be utilized as investment suggestions of any type. This report represents an extensive evaluation process designed to help our customers improve code quality while mitigating the high risks in blockchain.

3.4 About Beosin

Beosin is the first institution in the world specializing in the construction of blockchain security ecosystem. The core team members are all professors, postdocs, PhDs, and Internet elites from world-renowned academic institutions. Beosin has more than 20 years of research in formal verification technology, trusted computing, mobile security and kernel security, with overseas experience in studying and collaborating in project research at well-known universities. Through the security audit and defense deployment of more than 2,000 smart contracts, over 50 public blockchains and wallets, and nearly 100 exchanges worldwide, Beosin has accumulated rich experience in security attack and defense of the blockchain field, and has developed several security products specifically for blockchain.

BEOSIN
Blockchain Security

Official Website
https://www.beosin.com

Telegram
https://t.me/beosin

Twitter
https://twitter.com/Beosin_com

Email
service@beosin.com