Cyberscope

Audit Report

DeGuard

November 2023

Network	BSC
Address	0x905df659b18702091a4458f8ee7ca0302869abed
Network	MATIC
Address	0xd3FB798F925A0820fB222140834C55361FC730aF
Network	FTM
Address	0x5478315C88dc4A5C31cC9127b4ecC6d55057fEBa
Network	ARBITRUM
Address	0x794Cc29583b301072c1f5946D210C24b6c5D3a70
Network	BASE
Address	0x11c113efB490FbAd0A998D3870DE6Dc94f229188
Network	LINEA
Address	0x11c113efB490FbAd0A998D3870DE6Dc94f229188
Audited by	© cyberscope

Table of Contents

Overview

The `DeGuardNFT` contract, integrates several advanced features, including ERC721Upgradeable, ERC721EnumerableUpgradeable, ERC721BurnableUpgradeable, and IMultiChainToken, along with the AsterizmClientUpgradeable contracts. This contract is designed to provide a robust and flexible platform for NFT (Non-Fungible Token) management, particularly focusing on cross-chain transfer capabilities.

Audit Scope

In the context of the `DeGuardNFT` contract, the `initializerLib` address and its associated functionalities are outside the current audit scope. This contract heavily relies on the `initializerLib` for its operations, particularly in validating cross-chain transfers. Since the `initializerLib` is not within the purview of this audit, its functionality, security, and reliability are assumed to be accurate and effective without direct examination. This assumption is a critical consideration in understanding the overall security posture of the `DeGuardNFT` contract.

Owner Functionality

A key aspect of this contract is the extensive control vested in the owner. The owner has the ability to update the currency rate, which is crucial for maintaining the value of transactions in line with real-world currency fluctuations. This is achieved through the `updateRate` function, where the owner can set the `TOKEN/USD` rate, ensuring that the NFTs' prices remain aligned with the market.

Additionally, the owner can manage the NFT plans offered by the contract. This includes the ability to add new plans, update existing ones, and remove plans as needed, using the `addPlan`, `updatePlan`, and `removePlan` functions respectively. Each plan is characterized by its price in USD and its duration in days, allowing for a variety of options to be offered to users. The owner's ability to manage these plans provides flexibility in catering to different user needs and market conditions.

BuyPlan Functionality

The `buyPlan` function of the `DeGuardNFT` contract, enabling users to purchase NFT plans. When a user invokes this function, they can select a plan based on its unique ID. The function calculates the cost of the plan in the native network token, considering the current exchange rate set by the owner. Users must send the exact amount of cryptocurrency corresponding to the plan's price for the transaction to be successful.

Upon purchase, the contract mints a new NFT representing the plan and assigns it to the buyer. This NFT contains details such as the plan's ID, start time, and end time, encapsulating the duration for which the plan is valid. The minting process is handled internally by the `_safeMint` function, ensuring that the NFT is securely transferred to the user's wallet.

A feature of the `buyPlan` function is that supports cross-chain transfers. Users have the option to specify if the plan purchased is designed for a cross-chain transfer. If so, the contract facilitates the transfer of the NFT to a different blockchain network, as specified by the user. This feature significantly enhances the utility of the NFTs minted by this contract, allowing them to be used across multiple blockchain networks.

Roles

Owner

The owner can interact with the following functions in the "DeGuardNFT" NFT contract:

- updateRate(uint _value)
- addPlan(uint _id, uint _priceInUSD, uint _daysRange)
- removePlan(uint index)
- updatePlan(uint index, uint _id, uint _priceInUSD, uint _daysRange)
- withdraw()

The owner can interact with the following functions in the "AsterizmClientUpgradeable" NFT contract:

- setExternalRelay(address _externalRelay)
- addSender(address _sender)
- removeSender(address _sender)

- addTrustedAddress(uint64 _chainId, uint _trustedAddress)
- addTrustedAddresses(uint64[] calldata _chainIds, uint[] calldata _trustedAddresses)
- removeTrustedAddress(uint64 _chainId)

Users

Users can interact with the following functions in the "DeGuardNFT" NFT contract:

- buyPlan(uint _plan, uint64 _dstChainId, address _to, bool _crosschain)
- tokensOfOwner(address _owner)
- getRange(address _owner, uint _id)
- isValid(address _owner, uint _id)
- crossChainTransfer(uint64 _dstChainId, address _from, address _to, uint _tokenId)

Review

Contract Name	DeGuardNFT
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Network	BSC
Explorer	https://bscscan.com/address/0x905df659b18702091a4458f8ee7ca0302869abed
Address	0x905df659b18702091a4458f8ee7ca0302869abed
Network	MATIC
Explorer	https://polygonscan.com/address/0xd3fb798f925a0820fb222140834c55361fc730af
Address	0xd3FB798F925A0820fB222140834C55361FC730aF
Network	FTM
Explorer	https://ftmscan.com/address/0x5478315c88dc4a5c31cc9127b4ecc6d55057feba
Address	0x5478315C88dc4A5C31cC9127b4ecC6d55057fEBa
Network	ARBITRUM
Explorer	https://arbiscan.io/address/0x794cc29583b301072c1f5946d210c24b6c5d3a70
Address	0x794Cc29583b301072c1f5946D210C24b6c5D3a70
Network	BASE

Explorer	https://basescan.org/address/0x11c113efb490fbad0a998d3870de6dc94f229188
Address	0x11c113efB490FbAd0A998D3870DE6Dc94f229188
Network	Linea
Explorer	https://lineascan.build/address/0x11c113efb490fbad0a998d3870de6dc94f229188
Address	0x11c113efB490FbAd0A998D3870DE6Dc94f229188

Audit Updates

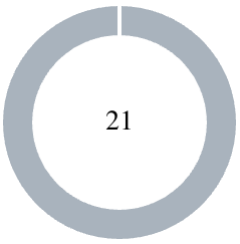
Initial Audit	22 Nov 2023

Source Files

Filename	SHA256
contracts/DeGuardNFT.sol	a4773896225ab63e7e1eb94216e6a4f9e6a7106fabfea3c27cc0850fc583079f
contracts/interfaces/IMultiChainToken.sol	e302211dc0302db730620b05747563d396a85ed6d6b13e58a43bb6787339a020
asterizmprotocol/contracts/evm/AsterizmClientUpgradeable.sol	deea6a6459f320e1ad8562be7e20a84d7e661503b731d61806d17e8e7f5b7476
asterizmprotocol/contracts/evm/libs/UintLib.sol	a4e46a027297f3e8a5663ac2e56d8ed120e4ec860c6f8a3340b43245bc2a543c
asterizmprotocol/contracts/evm/libs/AsterizmHashLib.sol	20a45afe5a613ff2970a1cd21bb962bedfc9e0a7f4c47d767b989746d1cc1117
asterizmprotocol/contracts/evm/libs/AddressLib.sol	67204a02478642ac9d056800ff1aa477f7b20f3f78e6b189c41c2403368bd648

asterizmprotocol/contracts/evm/interfaces/IInitializerSender.sol	63c1d026cb4480edbdd3f88ea55d1351e3c5983f6c7da63d5364f975e314f354
asterizmprotocol/contracts/evm/interfaces/IClientReceiverContract.sol	0e485b2969f80efeecde82ae66e35d83d4fe8dedce0a3551b6bf18cb5b41fe55
asterizmprotocol/contracts/evm/interfaces/IAsterizmEnv.sol	61e330f7b8540dd6b207c37a67b2bc8fbc787ac8c3ea817aa83027f03d7055b0
asterizmprotocol/contracts/evm/base/AsterizmEnv.sol	25340b3c6f6f1b28705606cea416a1e4b5d9f01361a8c7f123adcde152e17eea

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	21

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	21	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	CCR	Contract Centralization Risk	Unresolved
	IRM	Inefficient Removal Method	Unresolved
	IMU	Inconsistent Modifier Usage	Unresolved
	ALM	Array Length Mismatch	Unresolved
	RSU	Redundant Struct Usage	Unresolved
	MC	Missing Check	Unresolved
	UNMS	Undeclared NFT Max Supply	Unresolved
	RM	Redundant Modifier	Unresolved
	RFD	Redundant Function Declarations	Unresolved
	SVR	Struct Variable Redundancy	Unresolved
	RIC	Redundant If Check	Unresolved
	CO	Code Optimization	Unresolved
	MU	Modifiers Usage	Unresolved
	RCS	Redundant Comment Segments	Unresolved

●	RES	Redundant Event Statement	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	contracts/DeGuardNFT.sol#L174,206 asterizmprotocol/contracts/evm/AsterizmClientUpgradeable.sol#L286
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

Specifically, the contract is designed to grant the owner extensive control over critical aspects of its functionality. Specifically, the owner has the authority to set and update the rate of the price, add or remove plans available for purchase, and add trusted addresses for cross-chain transfers. While this centralized control might be intended for administrative convenience and flexibility, it also introduces significant risks.

```
    function updateRate(uint _value) public onlyOwner {
        require(_value > 0, "Currency rate must be higher than
zero");
        rate.value = _value;
        rate.updated = block.timestamp;
        emit RateUpdated(rate.value);
    }

    function addPlan(
        uint _id,
        uint _priceInUSD,
        uint _daysRange
        // string memory uri
    ) public onlyOwner {
        plans.push(Plan(_id, _priceInUSD, _daysRange * _days));

        emit PlanUpdated(plans.length - 1, _id, _priceInUSD,
_daysRange);
    }

    function removePlan(uint index) public onlyOwner {
        ...
    }

    function addTrustedAddress(uint64 _chainId, uint _trustedAddress)
public onlyOwner {
        trustedAddresses[_chainId].exists = true;
        trustedAddresses[_chainId].trustedAddress = _trustedAddress;
        trustedAddresses[_chainId].chainType =
initializerLib.getChainType(_chainId);

        emit AddTrustedAddressEvent(_chainId, _trustedAddress);
    }
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

IRM - Inefficient Removal Method

Criticality	Minor / Informative
Location	contracts/DeGuardNFT.sol#L233
Status	Unresolved

Description

The contract is utilizing a method for removing elements from the plans array. Specifically, the `removePlan` function employs a for loop to iterate through the array elements, shifting each element down by one index to remove the specified element. This approach, while functional, is not optimal in terms of gas usage and execution time, especially as the size of the array grows.

```
    function removePlan(uint index) public onlyOwner {
        require(index < plans.length, "Index is out of
bounds");

        Plan memory _plan = plans[index];

        for(uint i = index; i < plans.length - 1; i++){
            plans[i] = plans[i+1];
        }
        plans.pop();

        emit PlanRemoved(plans.length + 1, _plan.id,
_plan.price, _plan.range);
    }
```

Recommendation

It is recommended to enhance the efficiency of the `removePlan` function by adopting a more gas-efficient approach. This can be achieved by swapping the last element of the plans array with the element intended for removal, and then calling the `.pop` method to remove the last element. This method significantly reduces the number of operations required, especially for large arrays, thereby optimizing gas costs and execution time.

IMU - Inconsistent Modifier Usage

Criticality	Minor / Informative
Location	asterizmprotocol/contracts/evm/AsterizmClientUpgradeable.sol#L459
Status	Unresolved

Description

The contract contains the `asterizmClReceive` function which subsequently calls the `_asterizmReceiveInternal` function. The `asterizmClReceive` function is guarded by the `onlySender` modifier, while `_asterizmReceiveInternal` employs the `onlyOwnerOrInitializer` modifier. This discrepancy implies that for the successful execution of the entire operation, the sender should be the owner or the initialized address, as per the requirements of the `onlyOwnerOrInitializer` modifier. This inconsistency in modifier usage can lead to scenarios where the `asterizmClReceive` function is accessible to a sender, but the subsequent internal function call fails due to the different modifier requirement. Such a situation can prevent the successful execution of the intended functionality, leading to potential operational issues within the contract.

```
    function asterizmClReceive(uint64 _srcChainId, uint _srcAddress,
uint _txId, bytes32 _transferHash, bytes calldata _payload) external
onlySender nonReentrant {
        ...
    }

    function _asterizmReceiveInternal(ClAsterizmReceiveRequestDto
memory _dto) private
    onlyOwnerOrInitializer
    ...
    {
        ...
    }
```

`

Recommendation

It is recommended to reconsider and align the modifier implementation according to the specific requirements of the function. If both functions are meant to be accessible under the same conditions, the modifiers should be consistent across both. This could involve either adjusting the `asterizmClReceive` function to use the `onlyOwnerOrInitializer` modifier or modifying the `_asterizmReceiveInternal` function to use the `onlySender` modifier, depending on the intended access control logic. Ensuring consistency in modifier usage will enhance the contract's reliability and prevent execution failures due to conflicting access requirements.

ALM - Array Length Mismatch

Criticality	Minor / Informative
Location	asterizmprotocol/contracts/evm/AsterizmClientUpgradeable.sol#L296
Status	Unresolved

Description

The contract is designed to handle the addition of multiple trusted addresses through the `addTrustedAddresses` function, which accepts the arrays `_chainIds` and `_trustedAddresses`. This function iterates over these arrays, adding each pair of chain ID and trusted address to the contract. However, the function does not explicitly check whether the lengths of the `_chainIds` and `_trustedAddresses` arrays are equal. This oversight could lead to scenarios where the lengths of these arrays differ, potentially causing out-of-bounds access if one array is shorter than the other. Such a situation could result in unexpected behavior or errors in the contract's execution, impacting its reliability and security.

```
    function addTrustedAddresses(uint64[] calldata _chainIds, uint[]
calldata _trustedAddresses) external onlyOwner {
        for (uint i = 0; i < _chainIds.length; i++) {
            addTrustedAddress(_chainIds[i], _trustedAddresses[i]);
        }
    }
```

Recommendation

It is recommended to add a validation check at the beginning of the `addTrustedAddresses` function to ensure that the lengths of the `_chainIds` and `_trustedAddresses` arrays are equal. This can be implemented as a simple conditional statement that compares the lengths of both arrays and reverts the transaction if they do not match. This precautionary measure will prevent out-of-bounds errors and ensure that each chain ID is paired with its corresponding trusted address, thereby maintaining the integrity and intended functionality of the contract.

RSU - Redundant Struct Usage

Criticality	Minor / Informative
Location	asterizmprotocol/contracts/evm/AsterizmClientUpgradeable.sol#L93
Status	Unresolved

Description

The current implementation of the `Sender` struct in the contract contains the single boolean field `exists`. In Solidity, structs are typically used to group related data when multiple fields are involved, providing a more organized and readable code structure. However, when a struct is composed of only a single field, as in this case, it is more efficient and straightforward to represent this data as a standalone variable.

```
    struct Sender {
        bool exists;
    }
```

Recommendation

It is recommended to evaluate the necessity and benefits of using a struct for the `Sender` data. If the `exists` field is the only piece of data required and there are no foreseeable additions of more fields to the `Sender` struct, converting it into a simple boolean variable will be more efficient. This change can simplify the contract, reduce gas costs associated with struct usage, and improve overall code readability.

MC - Missing Check

Criticality	Minor / Informative
Location	asterizmprotocol/contracts/evm/AsterizmClientUpgradeable.sol#L285
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

The contract includes the `addTrustedAddress` function, which empowers the owner to set a trusted addresses. However, the function does not prevent the owner from setting the zero address as a trusted address. This omission allows a cross-chain transfers being directed to the zero address. Such transfers to the zero address are effectively equivalent to burning the tokens, as the zero address is a non-recoverable location in the blockchain ecosystem. This could lead to unintentional loss of tokens and could not reflect the intended functionality and of the contract.

```
    function addTrustedAddress(uint64 _chainId, uint _trustedAddress)
public onlyOwner {
        trustedAddresses[_chainId].exists = true;
        trustedAddresses[_chainId].trustedAddress = _trustedAddress;
        trustedAddresses[_chainId].chainType =
initializerLib.getChainType(_chainId);

        emit AddTrustedAddressEvent(_chainId, _trustedAddress);
    }
```

Recommendation

It is recommended to incorporate an additional check within the `addTrustedAddress` function to explicitly prevent the setting of the zero address as a trusted address. This can be achieved by adding a condition that validates the `_trustedAddress` parameter against the zero address before proceeding with the rest of the function's logic. Implementing this safeguard will enhance the contract's security by ensuring that trusted

addresses are valid and prevent the accidental burning of tokens through transfers to the zero address.

UNMS - Undeclared NFT Max Supply

Criticality	Minor / Informative
Location	contracts/DeGuardNFT.sol#L19,183
Status	Unresolved

Description

The contract implements a NFT functionality derived from the ERC721 standard. However, the contract does not declare a maximum supply for the NFTs. This omission is significant in the context of the `buyPlan` function, which includes the capability to mint new NFTs. Without a defined maximum supply, the `buyPlan` function can mint a number of NFTs without any restriction, potentially leading to oversupply issues. This lack of a supply cap could affect the rarity and value of the NFTs, and not align with typical NFT implementation where defining a maximum supply is the typical fundamental aspect of an ERC721 token.

```
contract DeGuardNFT is ERC721Upgradeable, ERC721EnumerableUpgradeable,
ERC721BurnableUpgradeable, IMultiChainToken, AsterizmClientUpgradeable {
...

    function buyPlan(uint _plan, uint64 _dstChainId, address _to, bool
_crosschain) public payable nonReentrant {
        address user = msg.sender;
        uint value = msg.value;
        require(rate.value > 0, "Currency rate must be higher than
zero");
        require(_plan < plans.length, "Index is out of bounds");
        uint price = (rate.value * plans[_plan].price) / 10 ** 18;
        require(value == price, "Not enough money sent");
        uint _currentTime = block.timestamp;
        uint _endTime = _currentTime + plans[_plan].range;
        // uint tokenId = safeMint(user, _currentTime, _endTime);
        uint256 tokenId = _tokenIdCounter.current();
        _tokenIdCounter.increment();
        _safeMint(user, tokenId);
        tokenToPlan[tokenId].id = plans[_plan].id;
        tokenToPlan[tokenId].startTime = _currentTime;
        tokenToPlan[tokenId].endTime = _endTime;
        if (_crosschain) crossChainTransfer(_dstChainId, user, _to,
tokenId);
        emit PlanSold(user, _plan, plans[_plan].id);
    }
...
}
```

Recommendation

It is recommended to reconsider the code implementation regarding the NFT max supply. If the intended functionality is to maintain alignment with the typical NFT logic, which often includes scarcity as a key element, then the contract should embody a maximum NFT supply. This can be achieved by adding a state variable to keep track of the total number of NFTs minted and checking against this cap in the `buyPlan` function before minting new NFTs. Implementing a max supply will help ensure that the contract adheres to standard NFT practices and maintains the intended economic properties of the NFTs.

RM - Redundant Modifier

Criticality	Minor / Informative
Location	asterizmprotocol/contracts/evm/AsterizmClientUpgradeable.sol#L149
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract declares the `onlySenderOrOwner` modifier. The modifier is not being used by the contract. As a result, the modifier is redundant.

```
    modifier onlySenderOrOwner {
        require(msg.sender == owner() ||
senders[msg.sender].exists, "AsterizmClient: only sender or
owner");
        _;
    }
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it. If the intended purpose of the code is to utilize the `onlySenderOrOwner` modifier, then the contract should incorporate it. Otherwise, since the modifier is not used within the contract, it can be safely removed to streamline the code.

RFD - Redundant Function Declarations

Criticality	Minor / Informative
Location	asterizmprotocol/contracts/evm/base/AsterizmEnv.sol
Status	Unresolved

Description

The `AsterizmEnv` contract, encompasses a variety of function implementations. However, only a subset of these functions, specifically the `_buildClInitTransferRequestDto`, `_buildIzIninTransferRequestDto`, and `_buildClAsterizmReceiveRequestDto`, are actively employed in the contract's implementation. This indicates that the remaining functions, despite being declared and defined, do not contribute to the contract's functionality as they are not invoked or utilized in any part of the contract. This situation leads to the presence of redundant functions within the `AsterizmEnv` contract, which do not serve any operational purpose.

```
abstract contract AsterizmEnv is IAsterizmEnv {

    function _buildBaseTransferDirectionDto(
        uint64 _srcChainId, uint _srcAddress,
        uint64 _dstChainId, uint _dstAddress
    ) internal pure returns(BaseTransferDirectionDto memory) {
        BaseTransferDirectionDto memory dto;
        dto.srcChainId = _srcChainId;
        dto.srcAddress = _srcAddress;
        dto.dstChainId = _dstChainId;
        dto.dstAddress = _dstAddress;

        return dto;
    }

    function _buildClInitTransferRequestDto(uint64 _dstChainId, uint
_dstAddress, uint _txId, bytes32 _transferHash, uint _feeAmount)
internal pure returns(ClInitTransferRequestDto memory) {
        ClInitTransferRequestDto memory dto;
        dto.dstChainId = _dstChainId;
        dto.dstAddress = _dstAddress;
        dto.transferHash = _transferHash;
        dto.feeAmount = _feeAmount;
        dto.txId = _txId;

        return dto;
    }

    ...
```

Recommendation

It is recommended to review and remove the redundant functions from the `AsterizmEnv` contract if they are not utilized in a meaningful way. This action would streamline the contract, eliminating unnecessary code and reducing potential confusion for developers and auditors. Simplifying the contract in this manner can also aid in maintaining clarity and focus on the contract's primary functionalities, thereby enhancing overall code quality and maintainability.

SVR - Struct Variable Redundancy

Criticality	Minor / Informative
Location	contracts/DeGuardNFT.sol#L49,197
Status	Unresolved

Description

The contract contains the `PurchasedPlan` struct that includes `startTime` and `endTime` variables. These variables are set during the execution of the `buyPlan` function. However, these variables, despite being assigned values, do not contribute to any meaningful functionality within the contract. Their presence does not influence or alter the contract's operations, as they are not utilized in any other part of the code. This lack of functional integration renders the `startTime` and `endTime` variables redundant within the current contract structure.

```
    struct Plan {
        uint id;
        uint price;
        uint range;
    }

    function buyPlan(uint _plan, uint64 _dstChainId, address _to,
bool _crosschain) public payable nonReentrant {
        ...
        tokenToPlan[tokenId].id = plans[_plan].id;
        tokenToPlan[tokenId].startTime = _currentTime;
        tokenToPlan[tokenId].endTime = _endTime;
        ...
    }
```

Recommendation

It is recommended to reassess the implementation of the `PurchasedPlan` struct, particularly concerning the `startTime` and `endTime` variables. If these variables do not serve a specific purpose or add value to the contract's functionality, it would be prudent to consider their removal. Eliminating unnecessary variables can streamline the contract,

reduce complexity, and potentially optimize gas usage, leading to a more efficient and focused contract design.

RIC - Redundant If Check

Criticality	Minor / Informative
Location	contracts/DeGuardNFT.sol#L307 contracts/ERC721BurnableUpgradeable.sol#L27
Status	Unresolved

Description

The contract is currently utilizing the `_debitFrom` function to burn a specific token ID. Within this function, there is an if statement designed to verify whether the caller has the necessary permission to burn the token. However, the internal `burn` function, which is called subsequently, already incorporates the same permission check. Specifically, the `burn` function includes a require statement that validates whether the caller is the token owner or is approved. This duplication of the permission check in both `_debitFrom` and `burn` functions results in unnecessary redundancy within the contract's logic.

```
    function _debitFrom(address _from, uint _tokenId) internal
virtual {
        address spender = _msgSender();
        if (_from != spender) _isApprovedOrOwner(spender,
_tokenId);
        burn(_tokenId);
    }

    function burn(uint256 tokenId) public virtual {
        require(_isApprovedOrOwner(_msgSender(), tokenId),
"ERC721: caller is not token owner or approved");
        _burn(tokenId);
    }
```

Recommendation

It is recommended to remove the `if` statement from the `_debitFrom` function, as the permission check is already effectively implemented within the internal `burn` function. Eliminating this redundant check can streamline the contract's code, reducing complexity and potential confusion. This simplification not only enhances the clarity and efficiency of the contract but also aligns with best practices in smart contract development by avoiding unnecessary operations.

CO - Code Optimization

Criticality	Minor / Informative
Location	contracts/DeGuardNFT.sol#L104,194
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

Specifically, the contract contains the `buyPlan` function which includes the same code functionality as the `safeMint` internal function. In the `buyPlan` function, the process of minting a token and assigning plan details to it is implemented directly within the function body. This implementation is essentially a duplication of what the `safeMint` function is designed to handle. Such duplication can increase the risk of inconsistencies and make the contract less efficient.

```
    function safeMint(address to, uint _id, uint _startTime, uint
_endTime) internal {
        uint256 tokenId = _tokenIdCounter.current();
        _tokenIdCounter.increment();
        tokenToPlan[tokenId].id = _id;
        tokenToPlan[tokenId].startTime = _startTime;
        tokenToPlan[tokenId].endTime = _endTime;
        _safeMint(to, tokenId);
    }
    ...
    function buyPlan(uint _plan, uint64 _dstChainId, address _to,
bool _crosschain) public payable nonReentrant {
        ...
        uint256 tokenId = _tokenIdCounter.current();
        _tokenIdCounter.increment();
        _safeMint(user, tokenId);
        tokenToPlan[tokenId].id = plans[_plan].id;
        tokenToPlan[tokenId].startTime = _currentTime;
        tokenToPlan[tokenId].endTime = _endTime;
        ...
    }
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it. It is advised to utilize the internal `safeMint` function within the `buyPlan` function instead of replicating its logic. By calling `safeMint` directly, the contract can achieve a more streamlined and efficient execution, reducing the size of the codebase.

MU - Modifiers Usage

Criticality	Minor / Informative
Location	contracts/DeGuardNFT.sol#L147,158,175,186,220,247 asterizmprotocol/contracts/evm/AsterizmClientUpgradeable.sol#L305,381,394
Status	Unresolved

Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

```
require(_id == tokenId, "Token does not belong to user");
require(rate.value > 0, "Currency rate must be higher than zero");
require(rate.value > 0, "Currency rate must be higher than zero");
require(index < plans.length, "Index is out of bounds");
require(trustedAddresses[_chainId].exists, "AsterizmClient: trusted
address not found");
require(trustedAddresses[_dstChainId].exists, "AsterizmClient: trusted
address not found");
```

Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

RCS - Redundant Comment Segments

Criticality	Minor / Informative
Location	contracts/DeGuardNFT.sol#L94,163,192,277
Status	Unresolved

Description

The contract contains multiple segments of code that are commented out. While commented code can serve as documentation or indicate future development plans, in this case, the commented-out code segments do not provide meaningful documentation or context. Instead, they introduce ambiguity regarding the intended features and current state of the contract. This could lead to confusion about the contract's capabilities and operational logic. The presence of such code may also suggest incomplete features or tentative updates that have not been fully implementated.

```
    // constructor(IInitializerSender _initializerLib)
    // ERC721("DeGuardPlan", "DGP")
    // AsterizmClient(_initializerLib, true, false)
    // {
    //     rate = Rate(0, block.timestamp);
    //     /// For testing ONLY
    //     // safeMint(msg.sender, block.timestamp, block.timestamp +
30 days);
    // }
...
    /// For some cases when owner wants to stop plan selling
    // function pause() public onlyOwner {
    //     _pause();
    // }
...
    // uint tokenId = safeMint(user, _currentTime, _endTime);
...
    // function unpause() public onlyOwner {
    //     _unpause();
    // }
...
    // uint256 tokenId = tokenOfOwnerByIndex(_from, _tokenId);
    // require(_tokenId == tokenId, "Token does not belong to user");
```

Recommendation

It is recommended to remove the segments of commented-out code from the contract. This will improve the clarity and readability of the contract's codebase, ensuring that it accurately reflects the implemented and active functionalities. Otherwise if certain commented-out segments are placeholders for future development, they should be replaced with clear documentation outlining the intended features.

RES - Redundant Event StatementDescription

Criticality	Minor / Informative
Location	contracts/DeGuardNFT.sol#L36
Status	Unresolved

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The `EncodedPayloadRecieved` event statement is not used in the contract's implemantation.

```
event EncodedPayloadRecieved(uint64 srcChainId, address
srcAddress, uint nonce, uint _transactionId, bytes payload);
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it. It is recommend removing the unused event statement from the contract..

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/DeGuardNFT.sol#L85,104,134,145,156,174,183,207,208,209,243,244,245,276 asterizmprotocol/contracts/evm/AsterizmClientUpgradeable.sol#L112,255,269,276,285,296,304,358,393,416,428,438,459
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
IInitializerSender _initializerLib
uint _endTime
uint _startTime
uint _id
address _owner
uint _value
uint _plan
bool _crosschain
address _to
uint64 _dstChainId
uint _priceInUSD
uint _daysRange
uint _tokenId
address _from

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	asterizmprotocol/contracts/evm/base/AsterizmEnv.sol#L14,51,69,88,124,170,197 asterizmprotocol/contracts/evm/AsterizmClientUpgradeable.sol#L482,487
Status	Unresolved

Description

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

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

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

```
function _buildBaseTransferDirectionDto(
        uint64 _srcChainId, uint _srcAddress,
        uint64 _dstChainId, uint _dstAddress
    ) internal pure returns(BaseTransferDirectionDto memory) {
        BaseTransferDirectionDto memory dto;
        dto.srcChainId = _srcChainId;
        dto.srcAddress = _srcAddress;
        dto.dstChainId = _dstChainId;
        dto.dstAddress = _dstAddress;

        return dto;
    }

...
```

Recommendation

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

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	asterizmprotocol/contracts/evm/base/AsterizmEnv.sol#L18,35,52,73,89,105,128,152,174,201
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
BaseTransferDirectionDto memory dto
ClInitTransferRequestDto memory dto
InternalClInitTransferRequestDto memory dto
TrSendMessageRequestDto memory dto
TrTransferMessageRequestDto memory dto
IzIninTransferRequestDto memory dto
IzAsterizmReceiveRequestDto memory dto
ClAsterizmReceiveRequestDto memory dto
IzReceivePayloadRequestDto memory dto
IzRetryPayloadRequestDto memory dto
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	contracts/DeGuardNFT.sol#L296
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
uint txId
```

Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	asterizmprotocol/contracts/evm/AsterizmClientUpgradeable.sol#L257
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
externalRelay = _externalRelay
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/DeGuardNFT.sol#L2 asterizmprotocol/contracts/evm/interfaces/IAsterizmEnv.sol#L2asterizmprotocol/contracts/evm/base/AsterizmEnv.sol#L2 asterizmprotocol/contracts/evm/AsterizmClientUpgradeable.sol#L2
Status	Unresolved

Description

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

```
pragma solidity ^0.8.19;
pragma solidity ^0.8.17;
```

Recommendation

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

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
DeGuardNFT	Implementation	ERC721Upgradeable, ERC721EnumerableUpgradeable, ERC721BurnableUpgradeable, IMultiChainToken, AsterizmClientUpgradeable		
	initialize	Public	✓	initializer
	safeMint	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	supportsInterface	Public		-
	tokensOfOwner	External		-
	getRange	External		-
	isValid	External		-
	updateRate	Public	✓	onlyOwner
	buyPlan	Public	Payable	nonReentrant
	addPlan	Public	✓	onlyOwner
	removePlan	Public	✓	onlyOwner
	updatePlan	Public	✓	onlyOwner
	getPlanList	Public		-

	withdraw	Public	✓	onlyOwner
	crossChainTransfer	Public	Payable	-
	_asterizmReceive	Internal	✓	
	_buildPackedPayload	Internal		
	_debitFrom	Internal	✓	
IMultiChainToken	Interface			
	crossChainTransfer	External	Payable	-
AsterizmClientUpgradeable	Implementation	UUPSUpgradeable, OwnableUpgradeable, ReentrancyGuardUpgradeable, IClientReceiverContract, AsterizmEnv		
	__AsterizmClientUpgradeable_init	Public	✓	initializer
	_authorizeUpgrade	Internal	✓	onlyOwner
	_setInitializer	Private	✓	
	_setLocalChainId	Private	✓	
	_setNotifyTransferSendingResult	Private	✓	
	_setDisableHashValidation	Private	✓	
	_getChainType	Internal		
	setExternalRelay	Public	✓	onlyOwner
	getExternalRelay	External		-
	addSender	Public	✓	onlyOwner

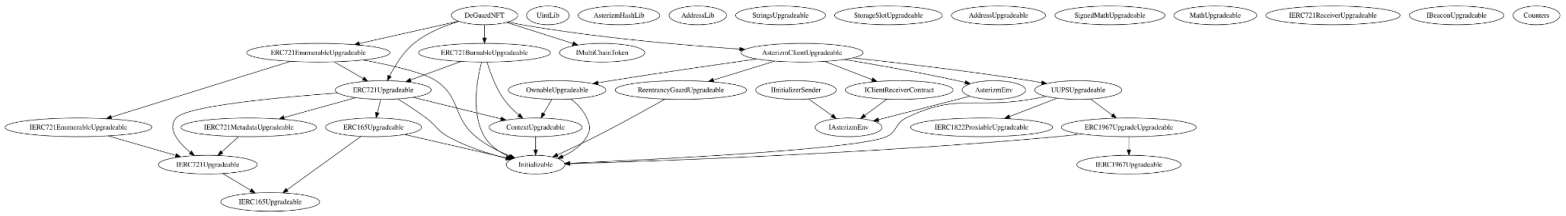
	removeSender	Public	✓	onlyOwner
	addTrustedAddress	Public	✓	onlyOwner
	addTrustedAddresses	External	✓	onlyOwner
	removeTrustedAddress	External	✓	onlyOwner
	_buildTransferHash	Internal		
	_validTransferHash	Internal		
	_getTxId	Internal		
	_getLocalChainId	Internal		
	getInitializerAddress	External		-
	getTrustedAddresses	External		-
	getDisableHashValidation	External		-
	getNotifyTransferSendingResult	External		-
	_initAsterizmTransferEvent	Internal	✓	
	initAsterizmTransfer	External	Payable	onlySender nonReentrant
	_initAsterizmTransferPrivate	Private	✓	onlyExistsOutboundTransfer onlyNotExecutedOutboundTransfer
	resendAsterizmTransfer	External	Payable	onlyOwner onlyExistsOutboundTransfer onlyExecutedOutboundTransfer
	transferSendingResultNotification	External	✓	onlyInitializer onlyExecutedOutboundTransfer
	asterizmIzReceive	External	✓	onlyInitializer

	_asterizmReceiveExternal	Private	✓	onlyOwnerOrInitializer onlyTrustedAddress onlyNonExecuted
	asterizmClReceive	External	✓	onlySender nonReentrant
	_asterizmReceiveInternal	Private	✓	onlyOwnerOrInitializer onlyReceivedTransfer onlyTrustedAddress onlyTrustedTransfer onlyNonExecuted onlyValidTransferHash
	_asterizmReceive	Internal	✓	
	_buildPackedPayload	Internal		
UintLib	Library			
	toAddress	Internal		
AsterizmHashLib	Library			
	buildSimpleHash	Internal		
	buildCrosschainHash	Internal		
AddressLib	Library			
	toUint	Internal		
	isContract	Internal		

IInitializerSender	Interface	IAsterizmEnv		
	initTransfer	External	Payable	-
	validIncomeTransferHash	External		-
	getLocalChainId	External		-
	getChainType	External		-
	resendTransfer	External	Payable	-
IClientReceiverContract	Interface	IAsterizmEnv		
	asterizmIzReceive	External	✓	-
	asterizmClReceive	External	✓	-
	transferSendingResultNotification	External	✓	-
IAsterizmEnv	Interface			
AsterizmEnv	Implementation	IAsterizmEnv		
	_buildBaseTransferDirectionDto	Internal		
	_buildClInitTransferRequestDto	Internal		
	_buildInternalClInitTransferRequestDto	Internal		
	_buildTrSendMessageRequestDto	Internal		
	_buildTrTarnsferMessageRequestDto	Internal		
	_buildIzIninTransferRequestDto	Internal		
	_buildIzAsterizmReceiveRequestDto	Internal		
	_buildClAsterizmReceiveRequestDto	Internal		

	_buildIzReceivePayloadRequestDto	Internal		
	_buildIzRetryPayloadRequestDto	Internal		

Inheritance Graph



Flow Graph

Proxy Contracts

Initial Audit, 22 Nov 2023

At the time of the audit report, the contracts with the following addresses is pointed out by the following proxy addresses in each network:

Network	Contract Address	Proxy Address
MATIC	https://polygonscan.com/address/0xd3fb798f925a0820fb222140834c55361fc730af	https://polygonscan.com/address/0xdb25309aaf93744a0883b44d1d3dfebed83b338b
BSC	https://bscscan.com/address/0x905df659b18702091a4458f8ee7ca0302869abed	https://bscscan.com/address/0x35a5206d4f58ae3114a356a18c0277dc032a17d5
FTM	https://ftmscan.com/address/0x5478315c88dc4a5c31cc9127b4ecc6d55057feba	https://ftmscan.com/address/0xdc85b66584e76d5eb75496b11d9c17f2d0432771
ARBITRUM	https://arbiscan.io/address/0x794cc29583b301072c1f5946d210c24b6c5d3a70	https://arbiscan.io/address/0x787cf93043fa92aae5cf7f1ce396c95e497f14dc
BASE	https://basescan.org/address/0x11c113efb490fbad0a998d3870de6dc94f229188	https://basescan.org/address/0xc3f81e786c4bcc6dd328c1f006fc57be3a5abfac
LINEA	https://lineascan.build/address/0x11c113efb490fbad0a998d3870de6dc94f229188	https://lineascan.build/address/0xc3f81e786c4bcc6dd328c1f006fc57be3a5abfac

Summary

The DeGuard contract implements a decentralized NFT-based system for managing cross-chain transfers. This audit examines the contract for potential security vulnerabilities, evaluates the business logic for robustness and efficiency, and suggests possible enhancements to ensure optimal functionality and safety in cross-chain interactions.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope’s prior written consent. This report is not nor should be considered an “endorsement” or “disapproval” of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope’s position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

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

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

The Cyberscope team

https://www.cyberscope.io