Security Audit

Date : 2024-06-11

QIE Domain Contracts

Auditor : Anjanay Raina

Executive Summary

Type	Smart Contract Audit
Audit Timeline	8 days
Runtime Environment	EVM
Languages	Solidity

Scope

Github Repository Link:

https://github.com/anjanayraina/QIE_domain_contracts

/contracts/ECDSAUpgradeable.sol

/contracts/ERC721.sol

/contracts/ERC721Enumerable.sol

/contracts/IMetadata.sol

/contracts/IRootRegistry.sol

/contracts/IZone.sol

/contracts/Metadata,sol

/contracts/Order.sol

/contracts/RootRegistry.sol

/contracts/Zone.sol

Summary of Findings

ID	Name	Description	Severity
H-01	‘=’ used in onlyRegistry modifier instead of ‘==’ causes faulty access control	Incorrect Access control on onlyRegistry	High
M-01	`OwnableUpgradeable` uses single-step ownership transfer	Single-step ownership transfer means that if a wrong address was passed when transferring ownership or admin rights it can mean that role is lost forever	Medium

M-02	Inconsistent State Management for RootRegistry::domainBurn( )	domainBurn() function doesn't update all the relevant data structures	
L–01	Uninitialized Owner Compromises Access Control	The ERC721 contract does not initialize the _owner variable within its constructor	Low
L-02	`abi.encodePacked()` should not be used with dynamic types when passing the result to a hash function such as `keccak256()`	Unless there is a compelling reason, abi.encode should be preferred	Low
G-01	Change function visibility from public to external	For all the public functions, the input parameters are copied to memory automatically, and it costs gas	Gas
G-02	Using `calldata` instead of `memory` for read-only arguments saves gas	Changing Function visibility can help save some gas on function calls	Gas
I-01	Over-Permissive Approval Mechanism Compromises Decentralization	The approve function in the ERC721 contract allows multiple actors, including	Informational

paymentAddress and
any address approved
for all tokens

Findings

[H-01] ‘=’ used in onlyRegistry modifier instead of ‘==’ causes faulty access control

Severity: High

Location: Zone.sol

Description:

In the Zone contract, the **`onlyRegistry`** modifier is intended to restrict function access to addresses listed in the **`registry`** mapping. However, there is a critical flaw in the modifier's implementation: it uses an assignment **(=)** instead of a comparison **(==)** operator in the access control check. This mistake not only allows unauthorized addresses to bypass the modifier's restrictions but also inadvertently modifies the **`registry`** mapping, potentially granting access to unauthorized addresses.

Impact:

1. **Unauthorized Access:** Functions protected by the **`onlyRegistry`** modifier can be called by any address, leading to potential unauthorized actions such as minting or burning tokens. This undermines the contract's security and integrity.
2. **Modification of Registry Mapping:** The incorrect use of the assignment operator (=) means that the **`registry`** mapping could be altered inadvertently, causing a random address to be added as an authorized registry. This further compromises the security of the contract.

3. **High-Security Risk:** Given that the **`onlyRegistry`** modifier protects critical functions like **`mint`** and **`burnDomain`**, any breach of this access control can lead to significant misuse and potential exploitation, impacting the overall trust and functionality of the protocol.

Recommendation
To fix this critical issue, the **onlyRegistry** modifier should be corrected to use the comparison operator (==) to properly check if the caller is an authorized registry address without modifying the registry mapping.

Proof of Concept

```
contract Metadata is OwnableUpgradeable,UUPSUpgradeable
```

[M-01] `OwnableUpgradeable` uses single-step ownership transfer

Severity: Medium

Location: Metadata.sol

Description: Single-step ownership transfer means that if a wrong address was passed when transferring ownership or admin rights it can mean that role is lost forever. The ownership pattern implementation for the protocol is in **Metadata.sol** where a single-step transfer is implemented.This can be a problem for all methods marked in **onlyOwner** throughout the protocol, some of which are core protocol functionality.

Impact
The emergency withdrawal functionality of the contract will be bricked meaning that no withdrawal can be made in case of a protocol failure or a hack.

Recommendation
It is a best practice to use a two-step ownership transfer pattern, meaning ownership transfer gets to a pending state and the new owner should claim his new rights, otherwise the old owner still has control of the contract. Consider using OpenZeppelin's **Metadata** contract

Proof of Concept

```
contract Metadata is OwnableUpgradeable,UUPSUpgradeable
```

[M-02] Inconsistent State Management for RootRegistry::domainBurn()

Severity: Medium

Location : RootRegistry.sol

Description: The **domainBurn** function in the RootRegistry contract is responsible for burning a domain. However, it does not fully update all relevant data structures. Specifically, the **mintedDomainNames**, **mintedDomainOwners** arrays, and the **existingDomain** mapping are not updated after a domain is burned. This oversight results in an inconsistent state within the contract, where a domain is considered burned but still appears in the arrays and mappings that track active domains.

Impact:

1. **Data Inconsistency:** The burned domain still exists in **`mintedDomainNames`**, **`mintedDomainOwners`**, and **`existingDomain`**, which causes discrepancies between the actual state of domains and the data structures maintaining their records.
2. **Operational Risks:** This inconsistency can lead to errors in domain management operations. For example, functions that rely on these data structures may incorrectly include burned domains, leading to unexpected behavior or incorrect data being returned or processed.

3. **User Confusion and Trust Issues:** Users may become confused if they see domains that have been burned still listed in domain registries or queries. This can erode trust in the protocol's reliability and data integrity.

Recommendation

To maintain consistent and reliable state management, the **`domainBurn`** function should be updated to remove the burned domain from all relevant data structures. Specifically, the function should ensure that the domain is removed from **`mintedDomainNames`**, **`mintedDomainOwners`**, and **`existingDomain`** mappings.

Consider adding a function like this that deletes the data from the arrays and the mapping :

```
function _removeBurnedDomainFromArrays(string memory label_, uint
_tokenId) private {
    // Find and remove the domain from mintedDomainNames and
mintedDomainOwners arrays
    for (uint i = 0; i < mintedDomainNames.length; i++) {
        if (keccak256(bytes(mintedDomainNames[i])) ==
keccak256(bytes(label_))) {
            // Swap the element to the end and pop it
            mintedDomainNames[i] =
mintedDomainNames[mintedDomainNames.length - 1];
            mintedDomainNames.pop();

            mintedDomainOwners[i] =
mintedDomainOwners[mintedDomainOwners.length - 1];
            mintedDomainOwners.pop();
            break;
        }
    }
}
```

Proof of Concept :

```
    function domainBurn(string calldata label_, string calldata zone_)
external{
        uint _tokenId = parentTokenId[label_];
        require(msg.sender == domainMetadata[_tokenId].owner,
'Registry:Access Denied');

        delete userDomain[domainMetadata[_tokenId].owner];
        domainMetadata[_tokenId].owner = address(0);
        domainMetadata[_tokenId].mintTime = block.timestamp;
        parentTokenId[label_] = 1;

        IZone(zones[zone_]).burnDomain(_tokenId);
        IMetadata(metadataAddress).clearMetadata(_tokenId);
        emit domainBurnt(_tokenId, label_);
    }
```

[L-01] Uninitialized Owner Variable Compromises Access Control in ERC721 Contract

Severity: Low

Location : ERC721.sol

Description: The ERC721 contract does not initialize the **_owner** variable within its constructor. Although **_owner** is declared and used in access control (onlyOwner modifier), it remains unset upon deployment. This can lead to severe access control vulnerabilities, where functions restricted to the owner could be improperly accessible or entirely unusable. Even though its initialized in the contracts that inherit it , any subsequent contracts that inherit it might not have the variable initialization leading to faulty access control

Impact

The **onlyOwner** modifier is rendered ineffective without **_owner** being set. This opens up critical functions to unauthorized access or prevents legitimate use if the inheriting contract does not set _owner.

Recommendation

To ensure secure and predictable ownership handling across all inheriting contracts, initialize **_owner** in the ERC721 constructor. This will provide a default owner (typically the deploying address) and prevent unintentional access control gaps.

Proof of Concept :

```
constructor(string memory name_, string memory symbol_) {
        _name = name_;
        _symbol = symbol_;
    }
```

[L-02]`abi.encodePacked()` should not be used with dynamic types when passing the result to a hash function such as `keccak256()`

Severity: Low

Location: RootRegistry.sol , ECDSAUpgradeable.sol , Zone.sol

Description
Use **abi.encode()** instead which will pad items to 32 bytes, which will prevent hash collisions (e.g. **abi.encodePacked(0x123,0x456)** => 0x123456 => **abi.encodePacked(0x1,0x23456)**, but **abi.encode(0x123,0x456)** => 0x0...1230...456). "Unless there is a compelling reason, **abi.encode** should be preferred". If there is only one argument to **abi.encodePacked()** it can often be cast to bytes() or bytes32() instead.

Recommendation

use abi.encode() instead of abi.encodePacked()

POC

```
    function _namehash(string memory label, string memory parent) private
pure returns(bytes32){
        return keccak256(abi.encodePacked(parent, label));
    }
```

[G-01] Change function visibility from public to external

Severity: Gas

Location : ERC721.sol , ERC721Enumerable.sol

Description
For all the public functions, the input parameters are copied to memory automatically, and it costs gas. If your function is only called externally, then you should explicitly mark it as external. External function's parameters are not copied into memory but are read from calldata directly. This small optimization in your solidity code can save you a lot of gas when the function input parameters are huge.

Recommendation
Change the visibility of the given functions to external to save some gas.

POC

```
  function name() public view virtual override returns (string memory) {
        return _name;
    }
```

[G-02] Using `calldata` instead of `memory` for read-only arguments saves gas

Severity: Gas

Location : ECDSAUpgradeable.sol, ERC721.sol, RootRegistry.sol, Zone.sol

Description
Use calldata for arguments that are not changed while the function calls.

Recommendation
Change memory to calldata in the function parameter

POC

```
    function tryRecover(bytes32 hash, bytes memory signature) internal
pure returns (address, RecoverError) {
        // Check the signature length
        // - case 65: r,s,v signature (standard)
        // - case 64: r,vs signature (cf
https://eips.ethereum.org/EIPS/eip-2098) _Available since v4.1._
        if (signature.length == 65) {
            bytes32 r;
            bytes32 s;
            uint8 v;
            // ecrecover takes the signature parameters, and the only way
to get them
            // currently is to use assembly.
            assembly {
                r := mload(add(signature, 0x20))
                s := mload(add(signature, 0x40))
                v := byte(0, mload(add(signature, 0x60)))
            }
            return tryRecover(hash, v, r, s);
        } else if (signature.length == 64) {
            bytes32 r;
            bytes32 vs;
            // ecrecover takes the signature parameters, and the only way
to get them
            // currently is to use assembly.
            assembly {
                r := mload(add(signature, 0x20))
                vs := mload(add(signature, 0x40))
            }
```

```
            return tryRecover(hash, r, vs);
        } else {
            return (address(0), RecoverError.InvalidSignatureLength);
        }
    }
```

[I-01] Over-Permissive Approval Mechanism Compromises Decentralization

Severity: Informational

Location: ERC721.sol

Description:

The approve function in the ERC721 contract allows multiple actors, including **paymentAddress** and any address approved for all tokens, to approve token transfers. While this design provides flexibility, it also poses a risk to decentralization. By permitting entities other than the token owner to approve transfers, the contract introduces a level of centralization that may deter users from participating in or investing in the protocol.

Impact:

1. **Risk to Decentralization:** Allowing **paymentAddress** and other non-owner addresses to approve token transfers undermines the principles of decentralization. This centralization could concentrate too much control in a single entity or a few entities, which is contrary to the ethos of decentralized finance (DeFi) and blockchain technology.
2. **User Trust and Participation:** Users might perceive the protocol as less trustworthy and more centralized, which can discourage investment and

participation. The ability for non-owners to approve transfers could be seen as a security risk or a potential point of manipulation, leading to reduced user confidence.

Recommendation

To align with decentralized principles and enhance user trust, it is recommended to limit the approval capability to the token owner only. Consider implementing a more decentralized approach, such as a voting system, where token holders can collectively decide on actions against bad actors or approve significant transfers. This would promote community governance and reduce the centralization risk.