QuillAudits

AUDIT REPORT

December 2025

For

INNOVATION CITY

Table of Content

Executive Summary

Project Name	Innovationcity
Protocol Type	Upgradeable ERC721 (UUPS) Soulbound Token
Project URL	https://innovationcity.com/
Overview	The DigitalLicense contract is a centrally controlled, revocable Soulbound Token (SBT) system. While it utilizes standard ERC-721 structures, it deviates significantly from "trustless" ownership models. The security of the system relies almost entirely on the operational security (OpSec) of the privileged roles - MINTER_ROLE, - UPGRADER_ROLE, - DEFAULT_ADMIN_ROLE.
Audit Scope	The scope of this Audit was to analyze the Innovationcity Smart Contracts for quality, security, and correctness.
Source Code link	https://github.com/rakdao-website/digital-license-contract
Branch	Main
Contracts in Scope	DigitalLicense.sol
Commit Hash	49d963966e8a664453269c916b45800951b495fa
Language	Solidity
Blockchain	Avalanche
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	3rd December 2025 - 8th December 2025
Updated Code Received	11th December 2025
Review 2	12th December 2025
Fixed In	https://github.com/rakdao-website/digital-license-contract/pull/2

Verify the Authenticity of Report on QuillAudits Leaderboard:

https://www.quillaudits.com/leaderboard

Number of Issues per Severity

Critical	0 (0.0%)
High	0 (0.0%)
Medium	3 (43.0%)
Low	0 (0.0%)
Informational	4 (57.0%)

Severity

Issues	Critical	High	Medium	Low	Informational
Open	0	0	0	0	0
Acknowledged	0	0	0	0	0
Partially Resolved	0	0	0	0	0
Resolved	0	0	**3**	0	**4**

Summary of Issues

Issue No.	Issue Title	Severity	Status
1	Inaccurate License Count After Revocation	Medium	Resolved
2	Tokens can be transferred even after expiration	Medium	Resolved
3	StartTime of 0 does not default to store current timestamp	Medium	Resolved
4	First Token ID Minted as Zero	Informational	Resolved
5	Misleading comment with roles assigned to revoke()	Informational	Resolved
6	Double event emission	Informational	Resolved
7	ERC-5192 Compliance Issue	Informational	Resolved

Checked Vulnerabilities

- Access Management
- Arbitrary write to storage
- Centralization of control
- Ether theft
- Improper or missing events
- Logical issues and flaws
- Arithmetic Computations Correctness
- Race conditions/front running
- SWC Registry
- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- ERC's conformance
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls

- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Multiple Sends
- Using suicide
- Using delegatecall
- Upgradeable safety
- Using throw
- Using inline assembly
- Style guide violation
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code
- Use of best practices
- Code documentation and comments, match logic and expected behavior
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper
- Efficient use of gas
- Code is safe from re-entrancy and other vulnerabilities

The following techniques, methods, and tools were used to review all the smart contracts:

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms Used for Audit

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity Static Analysis.

Types of Severity

Every issue in this report has been assigned to a severity level. There are five levels of severity, and each of them has been explained below.

Critical: Immediate and Catastrophic Impact

Critical issues are the ones that an attacker could exploit with relative ease, potentially leading to an immediate and complete loss of user funds, a total takeover of the protocol's functionality, or other catastrophic failures. Critical vulnerabilities are non-negotiable; they absolutely must be fixed.

High (H): Significant Risk of Major Loss or Compromise

High-severity issues represent serious weaknesses that could result in significant financial losses for users, major malfunctions within the protocol, or substantial compromise of its intended operations. While exploiting these vulnerabilities might require specific conditions to be met or a moderate level of technical skill, the potential damage is considerable. These findings are critical and should be addressed and resolved thoroughly before the contract is put into the Mainnet.

Medium (M): Potential for Moderate Harm Under Specific Circumstances

Medium-severity bugs are loopholes in the protocol that could lead to moderate financial losses or partial disruptions of the protocol's intended behavior. However, exploiting these vulnerabilities typically requires more specific and less common conditions to occur, and the overall impact is generally lower compared to high or critical issues. While not as immediately threatening, it's still highly recommended to address these findings to enhance the contract's robustness and prevent potential problems down the line.

Low (L): Minor Imperfections with Limited Repercussions

Low-severity issues are essentially minor imperfections in the smart contract that have a limited impact on user funds or the core functionality of the protocol. Exploiting these would usually require very specific and unlikely scenarios and would yield minimal gain for an attacker. While these findings don't pose an immediate threat, addressing them when feasible can contribute to a more polished and well-maintained codebase.

Informational (I): Opportunities for Improvement, Not Immediate Risks

Informational findings aren't security vulnerabilities in the traditional sense. Instead, they highlight areas related to the clarity and efficiency of the code, gas optimization, the quality of documentation, or adherence to best development practices. These findings don't represent any immediate risk to the security or functionality of the contract but offer valuable insights for improving its overall quality and maintainability. Addressing these is optional but often beneficial for long-term health and clarity.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Severity Matrix

Impact

Likelihood	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

Impact

- **High** - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- **Low** - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

Medium Severity Issues

Inaccurate License Count After Revocation

Resolved

Path

DigitalLicense.sol

Function Name

`getTotalLicenses(), revoke(), burn()`

Description

The getTotalLicenses() function returns misleading data after licenses are revoked because _nextTokenId is never decremented when tokens are burned, causing the total count to include revoked licenses.

Vulnerability Details

The contract uses _nextTokenId as an auto-incrementing counter for issuing new licenses and as the source of truth for the total license count:

```
function getTotalLicenses() external view returns (uint256) {
    return _nextTokenId;
}
```

However, when licenses are revoked through the revoke() function, the token is burned but _nextTokenId remains unchanged

```
function revoke(uint256 tokenId) external onlyRole(MINTER_ROLE) {
    if (_ownerOf(tokenId) == address(0)) revert NoToken();

    emit LicenseRevoked(tokenId, msg.sender);

    delete _tokenStartTime[tokenId];
    delete _tokenEndTime[tokenId];

    address owner = _ownerOf(tokenId);
    _approve(msg.sender, tokenId, owner);

    burn(tokenId); // Burns the token but _nextTokenId stays the same
}
```

Considering expired tokens will be different from burnt tokens as they can still be set back to unexpired however tokens burnt are totally not longer in existence therefore total number of license is bound to receive incorrect supply data which gets worse long term if current implementation gets kept.

Impact

- Off-chain systems relying on this value (dashboards, reporting tools, partner integrations, license validators) will receive incorrect supply data
- Total number of license is bound to receive incorrect supply data which gets worse long term if current implementation gets kept

Likelihood

Medium

Recommendation

Implement a separate counter to track accurate licenses supply and not _nextTokenId, as _nextTokenId can still be used for general mint count.

Tokens can be transferred even after expiration

Resolved

Path

DigitalLicense.sol

Function Name

`_update(), reassign()`

Description

The _update() hook enforces soulbound restrictions but does not check token expiration status. This allows MINTER_ROLE to transfer or reassign tokens that have expired:

```
function _update(
    address to,
    uint256 tokenId,
    address auth
) internal override(ERC721Upgradeable, ERC721PausableUpgradeable) returns
(address) {
    if (_ownerOf(tokenId) != address(0) && !hasRole(MINTER_ROLE, auth)) {
        if (to == address(0)) revert SoulboundBurn();
        revert SoulboundTransfer();
    }
    return super._update(to, tokenId, auth);
    // No expiration check - expired tokens can be transferred
}
```

An expired token can be reassigned via:

```
function reassign(uint256 tokenId, address newOwner, string calldata reason)
external onlyRole(MINTER_ROLE) {
    if (_ownerOf(tokenId) == address(0)) revert NoToken();
    if (newOwner == address(0)) revert ZeroAddress();
    // No expiration check

    // … proceeds to transfer expired token
}
```

Impact - HIGH

Violation of expiry semantics: Expired licenses should be invalid and non-transferable, but MINTER_ROLE can move them freely.
Regulatory compliance risk: If licenses represent legal permissions, transferring expired licenses violates their validity.

Likelihood - LOW

Requires MINTER_ROLE to intentionally or accidentally transfer an expired token. While not automatic, operational errors or malicious actions make this plausible.

Recommendation

Add expiration checks to _update() and reassign():

StartTime of 0 does not default to store current timestamp

Resolved

Path

DigitalLicense.sol

Function Name

`safeMint()`

Description

When minting tokens and startTimeVal is set to 0 as the current timestamp to mean that the token is 'live' immediately, the logs kept in the events as well as the storage slot for startTime value holds 0 and not the timestamp required.

If a dashboard intends to display any data related to the start time (e.g. how long a license has been in existence for, what months had the most licenses minted etc), this will return incorrect data for as many tokens that are minted with the startTimeVal set to 0.

Impact

Incorrect start time reporting: External systems querying startTime() receive 0 (Unix epoch: Jan 1, 1970) instead of the actual mint time
Broken time-based logic: Any logic depending on accurate start times (e.g., "token valid for X days from start") will malfunction

Likelihood

High - The default usage pattern will be startTimeVal = 0 for most mints (tokens that start immediately), affecting the majority of tokens.

Recommendation

If (1) startTime is very sensitive for data provision AND (2) there is no plan to migrate old licenses to this new system, there should be sanity checks that startTimeVal is not below the current timestamp so new licenses cannot be backdated.

Informational Issues

First Token ID Minted as Zero

Resolved

Path
DigitalLicense.sol

Function Name
`safeMint()`

Description
Mints begin with the tokenId at 0 which is contrary to what is specified in the docs here.

Impact
Low - This does not break any existing functionality.

Likelihood
High - Always happens with every deployed version of this contract.

Recommendation
Have the implementation contracts align with the specs as described.

Misleading comment with roles assigned to revoke()

Resolved

Path

DigitalLicense.sol

Function Name

`revoke()`

Description

Additionally, the contract's design document specifies that DEFAULT_ADMIN_ROLE should only manage roles, yet setContractURI() uses this role for contract metadata management.

Impact

Developer Confusion: Maintainers reading comments may misunderstand access control

Likelihood

Low

Recommendation

Update the NatSpec comments to accurately reflect the implementation.
Additionally, clarify the intended scope of DEFAULT_ADMIN_ROLE in the contract documentation.

Double event emission

Resolved

Path

DigitalLicense.sol

Function Name

`updateLicenseMetadata()`

Description

The updateLicenseMetadata() function emits a MetadataUpdate event after calling _setTokenURI(), which already emits the same event internally:

```
function updateLicenseMetadata(uint256 tokenId, string calldata newUri)
external onlyRole(MINTER_ROLE) {
    if (_ownerOf(tokenId) == address(0)) revert NoToken();

    if (bytes(newUri).length > 0) {
        _setTokenURI(tokenId, newUri); // Emits MetadataUpdate(tokenId)
        emit MetadataUpdate(tokenId);  // Duplicate emission
    }
}
```

Impact

Bloated logs: Unnecessary duplicate events increase transaction gas costs and blockchain storage

Likelihood

High - Occurs on every call to updateLicenseMetadata().

Recommendation

Remove the duplicate event emission.

ERC-5192 Compliance Issue

Resolved

Path

DigitalLicense.sol

Description

The locked() function does not follow ERC-5192 specification regarding non-existent tokens. Per the standard, it should revert for non-existent tokens, but currently returns false:

```
function locked(uint256 tokenId) external view override returns (bool) {
    return _ownerOf(tokenId) != address(0);
    // Should revert if token doesn't exist, not return false
}
```

Impact

High

Likelihood

High

Recommendation

Align the current implementation to the ERC-5192 specifications.

Centralization Risk

The above audit works under the assumption that all the admin roles will not act malicious under given circumstances.

The contract includes privileged administrative functions and ownership-controlled roles in: pause(), unpause(), mint(), _authorizeUpgrade(), setTokenExpiration() e.t.c. A misuse can pause tokens, prevent entire business logic, or mint arbitrary tokens

A single compromised or malicious MINTER_ROLE account can:

• Mint unlimited tokens to any address (safeMint)
• Burn any user's token (revoke, burn)
• Transfer tokens between arbitrary addresses (reassign)
• Update token metadata arbitrarily (updateLicenseMetadata)
• Modify token expiration dates (setTokenExpiration)

This creates a single point of failure where the MINTER_ROLE effectively has god-mode privileges over all tokens and user assets.

Roles should be assigned to multisig/timelock and add observability.

Functional Tests

Some of the tests performed are mentioned below:

- ✓ MINTER can revoke/burn and the token becomes non–existent.
- ✓ querying tokenURI on burned token reverts.
- ✓ total license count doesn't decrement after burning.
- ✓ empty-string updates are ignored, URIs can't be cleared.
- ✓ records logs show MetadataUpdate fires twice per metadata change.
- ✓ holders lacking MINTER_ROLE cannot burn.
- ✓ DEFAULT_ADMIN_ROLE alone can't revoke tokens.
- ✓ MINTER can reassign tokens without holder approval.
- ✓ reverse-ordered start/end times revert with InvalidTimeRange.
- ✓ direct holder transfers revert with SoulboundTransfer.
- ✓ pausing blocks MINTER-led reassignments.
- ✓ Unlocked event never emits during revocation.
- ✗ locked() reverts for non-existent IDs.
- ✗ mint with startTime 0 leaves stores start time as current timestamp
- ✗ expired tokens cannot be reassigned by MINTER.

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Threat Model

Contract	Function	Threats
DigitalLicense	safeMint	Unauthorized minting if MINTER_ROLE is compromised; metadata injection risk if malicious URI is supplied; issuance of licenses with unintended expiry ranges.
DigitalLicense	revoke	Malicious revocation and burning of valid user licenses if MINTER_ROLE is compromised; denial-of-service on holders.
DigitalLicense	reassign	Forced transfer of soulbound tokens if MINTER_ROLE is compromised; unauthorized identity reassignment; impersonation attack.
DigitalLicense	updateLicense Metadata	Metadata tampering leading to fraud/ misinformation if MINTER_ROLE is compromised.
DigitalLicense	setTokenExpiration	Malicious expiry changes (instantly expiring user licenses or extending them indefinitely) if MINTER_ROLE is compromised.
DigitalLicense	pause / unpause	System-wide denial of service if PAUSER_ROLE is misused or compromised.
DigitalLicense	setContractURI	Contract-level metadata manipulation; brand or policy misrepresentation if admin is compromised.
DigitalLicense	_authorizeUpgrade	Full system takeover via malicious upgrade implementation if UPGRADER_ROLE is compromised.
DigitalLicense	_update	Soulbound bypass attempts by external actors (blocked, but remains a threat surface).

Contract	Function	Threats
DigitalLicense	burn	Permanent deletion of licenses if MINTER_ROLE is misused or compromised.
DigitalLicense	supportsInterface	No direct threat, but integrators may misinterpret supported interfaces leading to unintended trust assumptions.
DigitalLicense	tokenURI	External metadata reliance risk; if URI host is compromised, users may receive malicious content.

Closing Summary

In this report, we have considered the security of Innovationcity Smart Contract. We performed our audit according to the procedure described above.

Issues of medium and informational severity were found, which have been resolved by the Innovationcity Team.

Disclaimer

At QuillAudits, we have spent years helping projects strengthen their smart contract security. However, security is not a one-time event—threats evolve, and so do attack vectors. Our audit provides a security assessment based on the best industry practices at the time of review, identifying known vulnerabilities in the received smart contract source code.

This report does not serve as a security guarantee, investment advice, or an endorsement of any platform. It reflects our findings based on the provided code at the time of analysis and may no longer be relevant after any modifications. The presence of an audit does not imply that the contract is free of vulnerabilities or fully secure.

While we have conducted a thorough review, security is an ongoing process. We strongly recommend multiple independent audits, continuous monitoring, and a public bug bounty program to enhance resilience against emerging threats.

Stay proactive. Stay secure.

About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With seven years of expertise, we've secured over 1400 projects globally, averting over $3 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

7+ Years of Expertise	**1M+** Lines of Code Audited
50+ Chains Supported	**1400+** Projects Secured

Follow Our Journey

AUDIT REPORT

December 2025

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com