





AUDIT REPORT

October 2025

For

complyexchange

Table of Content

Executive Summary	03
Number of Security Issues per Severity	05
Summary of Issues	06
Checked Vulnerabilities	07
Techniques and Methods	09
Types of Severity	11
Types of Issues	12
Severity Matrix	13
 Medium Severity Issues	14
1. Contract Owner Can Arbitrarily Destroy and Control All User Assets	14
 Low Severity Issues	15
2. Floating and Outdated Pragma	15
3. Inconsistent event semantics and missing ERC-4906 notifications on state-changing operations.	16
4. Use Ownable2Step version rather than Ownable version	17
Functional Tests	18
Automated Tests	19
Threat Model	20
Closing Summary & Disclaimer	21



Executive Summary

Project Name	FormTokenisation
Project URL	https://www.complyexchange.com/
Overview	<p>FormTokenisation is an upgradeable ERC721 NFT contract that severely restricts token transferability - only allowing minting and burning operations while blocking all standard transfers, approvals, and P2P transactions. The contract is designed as a "soul-bound" or non-transferable token system where only the owner can mint tokens (individually or in batches) and burn them through admin functions. It implements OpenZeppelin's upgradeable pattern with UUPS proxy support, reentrancy protection, and custom storage optimization using assembly for gas-efficient batch operations. Access to token metadata and ownership information is restricted to the contract owner only, making this unsuitable for public NFT marketplaces. The contract disables self-burning by token holders and all approval mechanisms, creating a centralized system where tokens can only exist within the original recipient's wallet. This design appears intended for certification, credential, or document tokenization use cases where transferability would undermine the token's purpose.</p>
Audit Scope	The scope of this Audit was to analyze the Solulabs Smart Contracts for quality, security, and correctness.
Source Code Link	https://github.com/SoluLab/ComplyExchange-SC
Branch	Main
Commit Hash	69a8ce95319c61ced4105c4a6872b920d5664768
Contracts in Scope	contracts/FormTokenisation.sol
Language	Solidity
Blockchain	EVM
Method	Manual Analysis, Functional Testing, Automated Testing



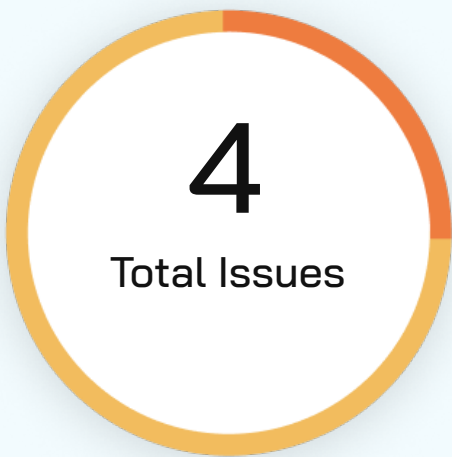
Review 1	27th August 2025 - 2nd September,2025
Updated Code Received	30th September 2025
Review 2	30th September 2025 - 8th October 2025
Fixed In	90a2a74d35b98568f8b87c1af4b5e360841b5ee8

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	1 (25.0%)
Low	3 (75.0%)
Informational	0 (0.0%)

		Severity				
		Critical	High	Medium	Low	Informational
Issues	Open	0	0	0	0	0
	Acknowledged	0	0	0	0	0
	Partially Resolved	0	0	0	0	0
	Resolved	0	0	1	3	0



Summary of Issues

Issue No.	Issue Title	Severity	Status
1	Contract Owner Can Arbitrarily Destroy and Control All User Assets	Medium	Resolved
2	Floating and Outdated Pragma	Low	Resolved
3	Inconsistent event semantics and missing ERC-4906 notifications on state-changing operations.	Low	Resolved
4	Use Ownable2Step version rather than Ownable version	Low	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
- Implementation of ERC standards
- 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		
		High	Medium	Low
Likelihood	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

Contract Owner Can Arbitrarily Destroy and Control All User Assets

Resolved

Path

contracts/FormTokenisation.sol

Path

`adminBurn()` , `batchAdminBurn()`

Description

The FormTokenisation contract exhibits severe centralization risks due to excessive administrative privileges granted to the contract owner. The vulnerability stems from the combination of several design decisions that concentrate unprecedented control in a single address. The contract owner possesses the ability to unilaterally burn any token through the `adminBurn` and `batchAdminBurn` functions without requiring consent from token holders. This is implemented through direct storage manipulation using assembly code that bypasses standard ERC721 safeguards, as seen in the `adminBurn` function where the contract directly clears the `_tokenURIs` storage slot and updates ownership to the zero address.

Additionally, the contract restricts access to fundamental ERC721 view functions such as `tokenURI` and `ownerOf` exclusively to the contract owner through the `onlyOwner` modifier. This creates an information asymmetry where token holders cannot independently verify their token ownership or metadata without relying on the owner's cooperation. The contract also disables all transfer functionality except for minting and burning, which while potentially intentional for the use case, further concentrates control by preventing users from moving their assets to safer custody arrangements.

Impact

The exploitation of these centralization risks could result in catastrophic financial losses for all token holders. A malicious or compromised owner could systematically destroy all tokens in existence using the batch burning functionality, effectively making all user investments worthless instantaneously.

Likelihood

Medium

Recommendation

The centralization risks require a comprehensive restructuring of the contract's governance and access control mechanisms. Implement a multi-signature wallet or decentralized autonomous organization structure for critical administrative functions, requiring multiple parties to approve destructive actions like token burning.



Low Severity Issues

Floating and Outdated Pragma

Resolved

Path

contracts/FormTokenisation.sol

Function Name

`pragma`

Description

Locking the pragma prevents the contract from being compiled with outdated Solidity versions that might contain security flaws.

In this case the pragma is left open as `>= 0.8.27`, meaning the code could be compiled with any later compiler release, which re-introduces the very risk the lock is meant to avoid.

Keep the compiler versions consistent in all the smart contract files. Do not allow floating pragmas anywhere. It is suggested to use the 0.8.29 pragma version

Inconsistent event semantics and missing ERC-4906 notifications on state-changing operations.

Resolved

Path

contracts/FormTokenisation.sol

Description

`batchAdminBurn()`

Description

The contract exposes two administrative burn paths with divergent event behavior. In `adminBurn(uint256 tokenId)`, the function clears the token URI storage via inline assembly, calls `_update(address(0), tokenId, tokenHolder)` to effect the burn, and emits only the custom `Burned(owner, tokenId)` event. It does not emit any ERC-4906 metadata event. In contrast, `batchAdminBurn(uint256[] calldata tokenIds)` clears URIs for each token, performs `_update(address(0), tokenId, tokenHolder)` for each burn, and then emits both `Burned(owner, tokenId)` and `MetadataUpdate(tokenId)` for every token.

This asymmetry means that a single-token burn executed via `adminBurn` does not produce the same event surface as an equivalent single token processed through the batch function.



Use Ownable2Step version rather than Ownable version**Resolved****Path**

contracts/FormTokenisation.sol

Function Name`constructor()`**Description**

Ownable2Step prevent the contract ownership from mistakenly being transferred to an address that cannot handle it (e.g. due to a typo in the address), by requiring that the recipient of the owner's permissions actively accept via a contract call of its own.

Consider using Ownable2Step from OpenZeppelin Contracts to enhance the security of your contract ownership management. This contract prevents the accidental transfer of ownership to an address that cannot handle it, such as due to a typo, by requiring the recipient of owner permissions to actively accept ownership via a contract call.



Functional Tests

Some of the tests performed are mentioned below:

- ✓ Contract deployment should set owner correctly
- ✓ Safe mint should assign token to correct user
- ✓ Safe mint should store the correct token URI
- ✓ Only owner should be able to mint new tokens
- ✓ Token balance should update correctly after minting
- ✓ Token ownership should transfer correctly between users
- ✓ Batch admin burn should remove multiple tokens successfully
- ✓ Batch admin burn should emit Burned and MetadataUpdate events
- ✓ Admin burn should remove single token successfully
- ✓ Admin burn should emit Burned event consistently
- ✓ Token URI should be cleared after burning a token
- ✓ Only owner should be able to perform admin burns
- ✓ Approve should allow another address to transfer the token
- ✓ setApprovalForAll should grant operator permissions correctly
- ✓ Transfer should update balances of sender and receiver correctly
- ✓ Safe transfer should preserve token URI after transfer
- ✓ Burned tokens should not be transferable anymore
- ✓ Querying ownerOf on burned token should revert
- ✓ Batch admin burn should handle empty array input gracefully
- ✓ Batch admin burn should fail for non-existent token IDs
- ✓ Multiple mints and burns should maintain consistent total supply



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
FormTokenisation	initialize	<ul style="list-style-type: none">- Centralization risk – single owner controls mint/burn/upgrade- Misconfiguration of initialOwner could lock contract
	safeMint	<ul style="list-style-type: none">- Owner abuse – mint unlimited tokens to self- Metadata injection (malicious URIs, phishing links)
	adminBurn	<ul style="list-style-type: none">- Inconsistent event emission vs batch burn (missing MetadataUpdate)- Assembly misuse in URI clearing could corrupt storage- Stealth deletion of tokens without off-chain detection
	batchAdminBurn	<ul style="list-style-type: none">- Gas griefing via large arrays- Potential denial if any token in batch doesn't exist



Closing Summary

In this report, we have considered the security of Solulabs's FormTokenisation. We performed our audit according to the procedure described above.

Issues of Medium and Low severity were found. Solulabs's FormTokenisation team resolved all the issues mentioned above.

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

October 2025

For

complyexchange



Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillaudits.com