TokenTable Suite Re-Audit

SECURITY ASSESSMENT REPORT

June 17, 2025

Prepared for

Contents

1 About CODESPECT

CODESPECT is a specialized smart contract security firm dedicated to ensure the safety, reliability, and success of blockchain projects. Our services include comprehensive smart contract audits, secure design and architecture consultancy, and smart contract development across leading blockchain platforms such as Ethereum (Solidity), Starknet (Cairo), and Solana (Rust).

At CODESPECT, we are committed to build secure, resilient blockchain infrastructures. We provide strategic guidance and technical expertise, working closely with our partners from concept development through deployment. Our team consists of blockchain security experts and seasoned engineers who apply the latest auditing and security methodologies to help prevent exploits and vulnerabilities in your smart contracts.

Smart Contract Auditing: Security is at the core of everything we do at CODESPECT. Our auditors conduct thorough security assessments of smart contracts written in Solidity, Cairo, and Rust, ensuring that they function as intended without vulnerabilities. We specialize in providing tailored security solutions for projects on EVM-compatible chains and Starknet. Our audit process is highly collaborative, keeping clients involved every step of the way to ensure transparency and security. Our team is also dedicated to cutting-edge research, ensuring that we stay ahead of emerging threats.

Secure Design & Architecture Consultancy: At CODESPECT, we believe that secure development begins at the design phase. Our consultancy services offer deep insights into secure smart contract architecture and blockchain system design, helping you build robust, secure, and scalable decentralized applications. Whether you're working with Ethereum, Starknet, or other blockchain platforms, our team helps you navigate the complexity of blockchain development with confidence.

Tailored Cybersecurity Solutions: CODESPECT offers specialized cybersecurity solutions designed to minimize risks associated with traditional attack vectors, such as phishing, social engineering, and Web2 vulnerabilities. Our solutions are crafted to address the unique security needs of blockchain-based applications, reducing exposure to attacks and ensuring that all aspects of the system are fortified.

With a focus on the intersection of security and innovation, CODESPECT strives to be a trusted partner for blockchain projects at every stage of development and for each aspect of security.

2 Disclaimer

Limitations of this Audit: This report is based solely on the materials and documentation provided to CODESPECT for the specific purpose of conducting the security review outlined in the Summary of Audit and Files. The findings presented in this report may not be comprehensive and may not identify all possible vulnerabilities. CODESPECT provides this review and report on an "as-is" and "as-available" basis. You acknowledge that your use of this report, including any associated services, products, protocols, platforms, content, and materials, is entirely at your own risk.

Inherent Risks of Blockchain Technology: Blockchain technology is still evolving and is inherently subject to unknown risks and vulnerabilities. This review focuses exclusively on the smart contract code provided and does not cover the compiler layer, underlying programming language elements beyond the reviewed code, or any other potential security risks that may exist outside of the code itself.

Purpose and Reliance of this Report: This report should not be viewed as an endorsement of any specific project or team, nor does it guarantee the absolute security of the audited smart contracts. Third parties should not rely on this report for any purpose, including making decisions related to investments or purchases.

Liability Disclaimer: To the maximum extent permitted by law, CODESPECT disclaims all liability for the contents of this report and any related services or products that arise from your use of it. This includes but is not limited to, implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Third-Party Products and Services: CODESPECT does not warrant, endorse, or assume responsibility for any third-party products or services mentioned in this report, including any open-source or third-party software, code, libraries, materials, or information that may be linked to, referenced by, or accessible through this report. CODESPECT is not responsible for monitoring any transactions between you and third-party providers. We strongly recommend conducting thorough due diligence and exercising caution when engaging with third-party products or services, just as you would for any other product or service transaction.

Further Recommendations: We advise clients to schedule a re-audit after any significant changes to the codebase to ensure ongoing security and reduce the risk of newly introduced vulnerabilities. Additionally, we recommend implementing a bug bounty program to incentivize external developers and security researchers to identify and disclose potential vulnerabilities safely and responsibly.

Disclaimer of Advice: FOR AVOIDANCE OF DOUBT, THIS REPORT, ITS CONTENT, AND ANY ASSOCIATED SERVICES OR MATERIALS SHOULD NOT BE CONSIDERED OR RELIED UPON AS FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER PROFESSIONAL ADVICE.

3 Risk Classification

Severity Level	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Table 1: Risk Classification Matrix based on Likelihood and Impact

3.1 Impact

- **High** - Results in a substantial loss of assets (more than 10%) within the protocol or causes significant disruption to the majority of users.
- **Medium** - Losses affect less than 10% globally or impact only a portion of users, but are still considered unacceptable.
- **Low** - Losses may be inconvenient but are manageable, typically involving issues like griefing attacks that can be easily resolved or minor inefficiencies such as gas costs.

3.2 Likelihood

- **High** - Very likely to occur, either easy to exploit or difficult but highly incentivized.
- **Medium** - Likely only under certain conditions or moderately incentivized.
- **Low** - Unlikely unless specific conditions are met, or there is little-to-no incentive for exploitation.

3.3 Action Required for Severity Levels

- **Critical** - Must be addressed immediately if already deployed.
- **High** - Must be resolved before deployment (or urgently if already deployed).
- **Medium** - It is recommended to fix.
- **Low** - Can be fixed if desired but is not crucial.

In addition to High, Medium, and Low severity levels, CODESPECT utilizes two other categories for findings: **Informational** and **Best Practices**.

a) **Informational** findings do not pose a direct security risk but provide useful information the audit team wants to communicate formally.

b) **Best Practices** findings indicate that certain portions of the code deviate from established smart contract development standards.

4 Executive Summary

This document presents the results of a security assessment conducted by CODESPECT for TokenTable. TokenTable is a token distribution platform that facilitates airdrops, vesting, and other mechanisms for distributing tokens.

This audit provides a focused review of previously audited EVM contracts, specifically evaluating the recent changes introduced by TokenTable. The primary areas of assessment include modifications to the deployment approach within the factory contract and the migration of key dependencies from OpenZeppelin to Solady libraries. Our objective was to ensure that these changes maintain the overall security and functionality of the system.

The audit was performed using:

a) Manual analysis of the codebase.

b) Dynamic analysis of programs, execution testing.

CODESPECT found two points of attention, two classified as `Best Practices`. All of the issues are summarised in Table 2.

Organisation of the document is as follows:

- **Section 5** summarizes the audit.
- **Section 6** describes the system overview.
- **Section 7** presents the issues.
- **Section 8** discusses the documentation provided by the client for this audit.
- **Section 9** presents the compilation and tests.

Acknowledgement of New Commit

CODESPECT acknowledges, through this audit report, a minor update to the `SIGNStaking` contract introduced in the following commit: 83429390fcf9f5ec8b74ef3da7ffaaead92f1970.

Issues found:

Severity	Unresolved	Fixed	Acknowledged
Best Practices	0	0	2
Total	**0**	**0**	**2**

Table 2: Summary of Unresolved, Fixed, and Acknowledged Issues

5 Audit Summary

Audit Type	Security Review
Project Name	TokenTable Suite
Type of Project	Token Distributors
Duration of Engagement	4 Days
Duration of Fix Review Phase	NaN
Draft Report	June 13, 2025
Final Report	June 17, 2025
Repository	tokentable-evm
Commit (Audit)	55dcbdea65d2fe0e3de515d1bfa5426200d17acc
Commit (Final)	55dcbdea65d2fe0e3de515d1bfa5426200d17acc
Documentation Assessment	Medium
Test Suite Assessment	Medium
Auditors	JecikPo, talfao

Table 3: Summary of the Audit

5.1 Scope - Audited Files

	Contract	LoC
1	TTADeployer.sol	49
2	TTUDeployer.sol	238
3	merkle/core/BaseMerkleDistributor.sol	269
4	merkle/core/extensions/SimpleNoMintERC721MerkleDistributor.sol	18
5	merkle/core/extensions/TokenTableNativeMerkleDistributor.sol	31
6	merkle/core/extensions/CustomFeesNativeMerkleDistributor.sol	98
7	merkle/core/extensions/TokenTableMerkleDistributor.sol	48
8	merkle/core/extensions/SimpleERC721MerkleDistributor.sol	17
9	merkle/core/extensions/custom/NFTGatedMerkleDistributor.sol	104
10	merkle/libs/delegatexyz/IDelegateRegistry.sol	133
11	merkle/utils/DelegateXYZProvider.sol	18
12	ecdsa/core/BaseECDSADistributor.sol	241
13	ecdsa/core/extensions/FungibleTokenWithFeesECDSADistributor.sol	102
14	ecdsa/core/extensions/FungibleTokenECDSADistributor.sol	64
15	common/core/TTFeeCollector.sol	79
16	common/interfaces/ITTFeeCollector.sol	16
17	common/utils/MinimalPausable.sol	45
18	common/utils/MinimalSafeERC20.sol	28
19	unlocker/core/TTUnlocker.sol	499
20	unlocker/core/TTTrackerToken.sol	55
21	unlocker/core/TTFutureToken.sol	94
22	unlocker/core/extensions/TTUnlockerNative.sol	62
23	unlocker/core/extensions/TTUnlockerExternalFT.sol	60
24	unlocker/proxy/TTUBeaconManager.sol	29
25	unlocker/interfaces/ITTUnlocker.sol	108
26	unlocker/interfaces/ITTFutureToken.sol	12
27	unlocker/interfaces/ITTTrackerToken.sol	5
28	unlocker/libs/IDelegateRegistry.sol	133
	Total	**2655**

5.2 Findings Overview

	Finding	Severity	Update
1	Non-standard owner initialization	Best Practices	Acknowledged
2	Unification of `SafeERC20` Imports	Best Practices	Acknowledged

6 System Overview

This audit covers incremental changes and additions to the existing TokenTable suite of smart contracts, which implement a flexible and extensible token distribution framework across EVM-compatible chains. The original system, already audited in prior engagements, supports signature- and Merkle proof–based claim validation for ERC20 and ERC721 assets, with configurable fee structures and claim hooks.

The current scope focuses on codebase refinements, modularization improvements, and the introduction of new functionality around the Unlocker deployment flow. The core contracts have undergone internal refactoring to replace OpenZeppelin (OZ) dependencies with lower-level or more gas-efficient Solady alternatives. These changes affect components such as initialization logic, ownership, reentrancy protection, merkle proof validation, and cloning mechanics.

In particular:

- **Deployment Contracts**: `TTADeployer` and `TTUDeployer` manage contract deployments. `TTUDeployer` is newly introduced to support the creation of Unlocker instances, optionally using the Beacon Proxy pattern for upgradability. Its `multicallDeploy(...)` enables atomic deployment and configuration with Presets and Actuals values.
- **Merkle-Based Distributors**: Several existing Merkle-based distribution contracts(`BaseMerkleDistributor`, `SimpleNoMintERC721MerkleDistributor`, etc.) now utilize Solady utilities in place of OpenZeppelin modules (e.g., `MerkleProofLib`, `Ownable`, `IERC721`). No functional changes were made to the core distribution logic.
- **ECDSA-Based Distributors**: Minor changes were applied to ECDSA-based contracts (`BaseECDSADistributor`, `FungibleTokenECDSADistributor`, `FungibleTokenWithFeesECDSADistributor`) to migrate from OpenZeppelin's `Initializable` and `SafeERC20` to Solady's counterparts or custom in-house utilities.
- **Token and Unlocker Logic**: Several token and unlocker contracts (`TTUnlocker`, `TTTrackerToken`, `TTFutureToken`) were adapted to use Solady's `Ownable` and `EnumerableSetLib`. The core logic remains unchanged.
- **New Components**: The audit also includes new lightweight modules such as `MinimalPausable`, `MinimalSafeERC20`, and `TTUBeaconManager`, which support enhanced configurability and simplified control over upgradable Unlocker deployments.

No changes were made to the fee logic (`TTFeeCollector`), native unlockers (`TTUnlockerNative`, `TTUnlockerExternalFT`), NFT gating (`NFTGatedMerkleDistributor`), or integration interfaces (`DelegateXYZProvider`), apart from dependency replacements where applicable.

7 Issues

7.1 [Best Practice] Non-standard owner initialization

File(s): `TTADeployer.sol` `TTUDeployer.sol`

Description: The `TTADeployer` and `TTUDeployer` contracts contain initialization function which sets the owner and can be called only once (by using the `initializer` modifier). In both cases, they use the `_setOwner(...)` function to set the owner.

```
function initialize(address owner) external initializer {
    _setOwner(owner);
}
```

While this doesn't introduce issues, the `_setOwner(...)` function does not verify if the owner was already initialized. It is recommended to use `_initializeOwner(...)` instead, to provide this extra check as suggested by the Solady best practices.

Impact: No security impact, best practice not followed.

Recommendation(s): Use `_initializeOwner(...)` to set the contract's owner.

Status: Acknowledged

Update from TokenTable: Acknowledged.

7.2 [Best Practice] Unification of `SafeERC20` Imports

File(s): `*.sol`

Description: The project currently uses two different implementations of the `SafeERC20` library across contracts: the OpenZeppelin (OZ) version and a custom minimal version (`MinimalSafeERC20`). This inconsistency is illustrated in the output below, e.g. the `TTUDeployer.sol` file includes:

```
import { SafeERC20 } from "@openzeppelin-contracts/token/ERC20/utils/SafeERC20.sol";
```

while the `FungibleTokenWithFeesECDSADistributor.sol` file contains:

```
import { SafeERC20 } from "../../../common/utils/MinimalSafeERC20.sol";
```

The following files use the OZ implementation:

- `TTUDeployer.sol`
- `TTFeeCollector.sol`
- `BaseECDSADistributor.sol`
- `BaseMerkleDistributor.sol`
- `TTUnlocker.sol`

Maintaining multiple versions of a commonly used utility library can introduce confusion, lead to inconsistent behavior, and increase maintenance complexity.

Impact: Standardising the import source improves code clarity and maintainability, which is a best practice concern.

Recommendation: Consider standardising the usage of `SafeERC20` across all contracts—either consistently use the OpenZeppelin implementation or the internal `MinimalSafeERC20`, depending on the project's performance or dependency goals.

Status: Acknowledged

Update from TokenTable:

8 Evaluation of Provided Documentation

The TokenTable documentation was provided in two forms:

- **Official Documentation Website:** The TokenTable Docs contain a high-level explanation of the Unlocker and its latest implemented changes, providing an overview of the protocol's purpose for both users and auditors.
- **Natspec Comments:** Some parts of the code included Natspec comments, which explained the purpose of complex functionality in detail and facilitated understanding of individual functions. However, some functionalities lacked comments, and expanding documentation coverage would enhance the overall comprehensibility of the code.

The documentation provided by TokenTable offered valuable insights into the protocol, significantly aiding CODESPECT's understanding. However, the public technical documentation could be further improved to better present the protocol's overall functionality and facilitate the understanding of each component.

Additionally, the TokenTable team was consistently available and responsive, promptly addressing all questions raised by CODESPECT during the evaluation process.

9 Test Suite Evaluation

9.1 Compilation Output

TT EVM Suite compilation output:

```
% forge build
[] Compiling...
[] Compiling 110 files with Solc 0.8.29
[] Solc 0.8.29 finished in 3.77s
Compiler run successful!
```

9.2 Tests Output

TT EVM Suite test output:

```
% forge test
[] Compiling...
No files changed, compilation skipped

Ran 5 tests for test/ecdsa/FungibleTokenWithFeesECDSADistributor.t.sol:FungibleTokenWithFeesECDSADistributorTest
[PASS] test_Claim_OutsideTimeRangeAndInactive() (gas: 349531)
[PASS] test_claim() (gas: 375391)
[PASS] test_claim_chargeFixedFee() (gas: 277932)
[PASS] test_decodeUserClaimData() (gas: 11536)
[PASS] test_getStorage() (gas: 37186)
Suite result: ok. 5 passed; 0 failed; 0 skipped; finished in 2.91ms (1.12ms CPU time)

Ran 14 tests for test/unlocker/TTUnlocker.t.sol:TTUnlockerTest
[PASS] test_Core_Initialize() (gas: 13233)
[PASS] test_Core_calculateClaimableAmount() (gas: 2136477)
[PASS] test_Core_createActualAndEnforcePermissions() (gas: 917345)
[PASS] test_Core_createPresetAndEnforcePermissions() (gas: 664916)
[PASS] test_Core_disableCreate() (gas: 585983)
[PASS] test_Core_founderCancelablesCancelAndRefund() (gas: 897941)
[PASS] test_Core_investorClaimCorrectAmount() (gas: 1083068)
[PASS] test_Core_withdrawDepositAndEnforcePermissions() (gas: 767317)
[PASS] test_DelegateClaim_NoRegistryNotPermissioned() (gas: 881981)
[PASS] test_Deployer_deployAndCompleteBeaconUpgrade() (gas: 2268140)
[PASS] test_Deployer_deployAsCloneAndNotUpgrade() (gas: 2175078)
[PASS] test_TrackerToken_IncludesPendingAmountsFromCancelledActuals() (gas: 898559)
[PASS] test_TrackerToken_balanceOfWithCancelledActual() (gas: 799454)
[PASS] test_TrackerToken_reflectCorrectClaimableAmount() (gas: 1118448)
Suite result: ok. 14 passed; 0 failed; 0 skipped; finished in 11.81ms (5.59ms CPU time)

Ran 8 tests for test/ecdsa/FungibleTokenECDSADistributor.t.sol:FungibleTokenECDSADistributorTest
[PASS] testFuzz_getClaimStatus(bytes32[]) (runs: 256, : 179441, ~: 186433)
[PASS] testFuzz_withdraw(uint256,uint256,bool) (runs: 256, : 81074, ~: 93447)
[PASS] test_claim() (gas: 435252)
[PASS] test_decodeUserClaimData() (gas: 10559)
[PASS] test_encodeHashToBeSigned() (gas: 12847)
[PASS] test_getStorage() (gas: 37095)
[PASS] test_setBaseParams() (gas: 54943)
[PASS] test_verify() (gas: 43246)
Suite result: ok. 8 passed; 0 failed; 0 skipped; finished in 54.62ms (63.32ms CPU time)

Ran 13 tests for test/common/TTFeeCollector.t.sol:TTUFeeCollectorTest
[PASS] testFuzz_getFee(address,uint256,uint16,uint128,uint256) (runs: 256, : 81864, ~: 85802)
[PASS] testFuzz_getFeeToken(address,address,address) (runs: 256, : 59831, ~: 59831)
[PASS] testFuzz_setCustomFeeBips_fail_feesTooHigh(address,uint256) (runs: 256, : 12411, ~: 12411)
[PASS] testFuzz_setCustomFeeBips_fail_notOwner(address,address,uint256) (runs: 256, : 12943, ~: 12943)
[PASS] testFuzz_setCustomFeeBips_success(address,uint256) (runs: 256, : 35506, ~: 36983)
[PASS] testFuzz_setCustomeFeeToken_success(address,address) (runs: 256, : 34402, ~: 34402)
[PASS] testFuzz_setDefaultFeeToken_success(address) (runs: 256, : 34033, ~: 34033)
[PASS] testFuzz_setDefaultFee_fail_notOwner(address,uint256) (runs: 256, : 11749, ~: 11749)
```

```
[PASS] testFuzz_setDefaultFee_success(uint256) (runs: 256, : 34555, ~: 34789)
[PASS] testFuzz_withdrawFee_fail_notOwner(address,uint256) (runs: 256, : 69115, ~: 69271)
[PASS] testFuzz_withdrawFee_fail_wrongToken(address,uint256) (runs: 256, : 71396, ~: 71883)
[PASS] testFuzz_withdrawFee_success_erc20(uint256) (runs: 256, : 85904, ~: 86531)
[PASS] testFuzz_withdrawFee_success_ether(uint128) (runs: 256, : 16123, ~: 16307)
Suite result: ok. 13 passed; 0 failed; 0 skipped; finished in 9.71s (203.26ms CPU time)

Ran 16 tests for test/merkle/TokenTableMerkleDistributor.t.sol:TokenTableMerkleDistributorTest
[PASS] testFuzz_claim_fail_notActive(uint128,uint128) (runs: 256, : 87823, ~: 88290)
[PASS] testFuzz_decodeLeaf(uint256,uint256,uint256) (runs: 256, : 13627, ~: 13627)
[PASS] testFuzz_delegateClaim_fail_notActive(address,uint128,uint128) (runs: 256, : 113115, ~: 113582)
[PASS] testFuzz_delegateClaim_fail_notDelegate(address,uint128,uint128) (runs: 256, : 112816, ~: 113361)
[PASS] testFuzz_encodeLeaf_verify_and_claim(address[],bytes32[],uint256[],uint64[],uint128[]) (runs: 256, : 28918921,
↪  ~: 27413641)
[PASS] testFuzz_encodeLeaf_verify_and_delegateClaim(address,address[],bytes32[],uint256[],uint64[],uint128[]) (runs:
↪  256, : 23836602, ~: 17876742)
[PASS] testFuzz_encodeLeaf_verify_no_claim(address[],bytes32[],uint256[],uint256[],uint256[]) (runs: 256, : 14340045,
↪  ~: 11987661)
[PASS] testFuzz_externalTTUFeeCollector_erc20Fees(address[],bytes32[],uint256[],uint64[],uint128[]) (runs: 256, :
↪  31612110, ~: 27604958)
[PASS] testFuzz_externalTTUFeeCollector_etherFees(address[],bytes32[],uint256[],uint64[],uint128[]) (runs: 256, :
↪  30448765, ~: 27464927)
[PASS] testFuzz_setBaseParams_fail_badTime(address,uint256,uint256,bytes32) (runs: 256, : 14417, ~: 14417)
[PASS] testFuzz_setBaseParams_fail_notOwner(address,address,uint256,uint256,bytes32) (runs: 256, : 16167, ~: 16167)
[PASS] testFuzz_setBaseParams_succeed_0(address,uint256,uint256,bytes32) (runs: 256, : 107062, ~: 107607)
[PASS] testFuzz_setClaimDelegate_fail_notOwner(address,address) (runs: 256, : 15795, ~: 15795)
[PASS] testFuzz_setClaimDelegate_succeed_0(address) (runs: 256, : 37906, ~: 37906)
[PASS] testFuzz_withdraw_fail_notOwner(address) (runs: 256, : 14839, ~: 14839)
[PASS] testFuzz_withdraw_succeed_0(uint256) (runs: 256, : 123958, ~: 124269)
Suite result: ok. 16 passed; 0 failed; 0 skipped; finished in 10.23s (45.75s CPU time)

Ran 4 tests for test/merkle/NFTGatedMerkleDistributor.t.sol:NFTGatedMerkleDistributorTest
[PASS] testFuzz_decodeMOCALeafData(uint256,uint256,uint256,uint256,uint256) (runs: 256, : 15371, ~: 15371)
[PASS] testFuzz_verifyAndClaim(address[],bytes32[],uint256[],uint8[],uint128[],uint8[]) (runs: 256, : 40414113, ~:
↪  38862369)
[PASS] test_disabledFunctions() (gas: 29546)
[PASS] test_setNft() (gas: 40784)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 10.97s (10.98s CPU time)

Ran 6 test suites in 10.98s (30.98s CPU time): 60 tests passed, 0 failed, 0 skipped (60 total tests)
```

9.3 Notes about Test suite

The TokenTable team test suite was already reviewed as part of the previous audits. The code changes that are in scope of this audit mostly don't require modification of the individual test cases. In case of the new deployment functionality that was added in `TTUDeployer` the coverage of the test cases was updated currently.

Yet, the test suite could be further enhanced by adding a scenario where the `TTUMulticallDeployer` contract is used to deploy the TTU Suite.

CODESPECT also recommends explicitly defining strict invariants that the protocol must uphold. Incorporating tests to validate these invariants would ensure that critical assumptions about the system's behaviour are consistently maintained across all functionalities, further bolstering the protocol's security and stability.