UnlockerV2

SECURITY ASSESSMENT REPORT

25 April, 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 focuses on the EVM-based Unlocker and Fee Collector contracts. The Unlocker contract is responsible for distributing tokens according to a predefined schedule, with flexible configuration options for how the tokens are released.

The audit was performed using:

a) Manual analysis of the codebase.

b) Dynamic analysis of programs, execution testing.

CODESPECT found ten points of attention, one classified as `Medium`, four classified as `Low`, four classified as `Informational` and one classified as `Best Practice`. 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.

Issues found:

Severity	Unresolved	Fixed	Acknowledged
Medium	0	1	0
Low	0	3	1
Informational	0	3	1
Best Practice	0	1	0
Total	**0**	**8**	**2**

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

5 Audit Summary

Audit Type	Security Review
Project Name	UnlockerV2
Type of Project	Token Distribution Platform
Duration of Engagement	10 Days
Duration of Fix Review Phase	1 Day
Draft Report	April 25, 2025
Final Report	April 25, 2025
Repository	tokentable-v2-evm
Commit (Audit)	e27192f627ea849f88e8a4b68382c5ac8808e3a5
Commit (Final)	da95751d63965671d35073347ca5ab69acf65599
Documentation Assessment	Medium
Test Suite Assessment	Medium
Auditors	Talfao, Kalogerone

Table 3: Summary of the Audit

5.1 Scope - Audited Files

	Contract	LoC
1	libraries/IDelegateRegistry.sol	89
2	libraries/CustomERC2771Context.sol	37
3	core/TokenTableUnlockerV2.sol	462
4	core/TTUFeeCollector.sol	80
5	core/TTUProjectTokenStorage.sol	28
6	core/TTTrackerTokenV2.sol	54
7	core/TTFutureTokenV2.sol	94
8	core/extensions/native-token/TTUV2Native.sol	59
9	core/extensions/ERC2771/TTUV2Gasless.sol	30
10	core/extensions/ERC2771/TTFTV2Gasless.sol	18
11	core/extensions/external-ft/TTUV2ExternalFT.sol	45
12	proxy/TTUMulticallDeployer.sol	61
13	proxy/TTUDeployerLite.sol	159
14	proxy/TTUV2BeaconManager.sol	19
15	interfaces/IVersionable.sol	4
16	interfaces/ITTHook.sol	4
17	interfaces/ITTTrackerTokenV2.sol	5
18	interfaces/ITTFutureTokenV2.sol	15
19	interfaces/ITokenTableUnlockerV2.sol	92
20	interfaces/ITTUFeeCollector.sol	15
21	interfaces/ITTUDeployer.sol	34
22	interfaces/TokenTableUnlockerV2DataModels.sol	14
23	interfaces/IOwnable.sol	6
	Total	**1424**

5.2 Findings Overview

	Finding	Severity	Update
1	Fixed fees allow users to transfer all the project tokens from the `Unlocker` to the protocol owner	Medium	Fixed
2	Tracker token's `balanceOf` will revert if address owns cancelled actuals	Low	Fixed
3	Tracker token's `totalSupply` is always incorrect	Low	Fixed
4	Unauthorized claims allowed when `externalDelegateRegistry` is not configured	Low	Fixed
5	`defaultFee` can be huge or insignificant depending on the project token's decimals	Low	Acknowledged
6	Deploying an `Unlocker` with Multicall may revert for 0 deposit amounts	Info	Acknowledged
7	Future token's `getClaimInfo` function can return inaccurate information about token's cancel status	Info	Fixed
8	Implementation of `CustomERC2771Context` based on the vulnerable version of OZ contract	Info	Fixed
9	Tracker token's `balanceOf` doesn't account for cancelled actuals	Info	Fixed
10	Order of calculation in `simulateAmountClaimable` function	Best Practices	Fixed

6 System Overview

TokenTable operates a token distribution platform that enables project owners to deploy customizable unlocker contracts for managing token vesting and allocations. Project owners define token allocations, which are distributed to users, who can then claim their tokens accordingly. TokenTable charges a fee on each claim, as governed by the fee collector contract.

6.1 Token Distribution/Unlocker Suite

To utilize the **TokenTable** platform, a project owner must deploy an instance of the `TokenTableUnlockerV2` contract via the `TTTUDeployerLite`. This deployment process also creates instances of the `TTFutureTokenV2` and `TTTrackerTokenV2` contracts.

Within the `TokenTableUnlockerV2`, the project owner configures token distributions using the `Preset` struct. Each `Preset` provides flexible configuration options, including the distribution start time, number of unlocks, and the timestamps for each unlock. After defining a `Preset`, the owner creates instances of the `Actual` struct, each representing a specific token allocation linked to a `Preset` and assigned to a particular user. The project owner must deposit the total amount of tokens to be distributed, along with the applicable fees, directly into the unlocker contract.

Each `Actual` instance stores data such as the token allocation and the associated `Preset` ID in the unlocker contract. Ownership of each `Actual` is represented by an NFT from the `TTFutureTokenV2` ERC-721 contract. During deployment, the project owner can optionally restrict the transferability of these NFTs.

Once the token distribution or vesting schedule is configured, users must wait according to the conditions set in the `Preset` before claiming their allocations. Users can claim their tokens via the `claim(...)` function, or an authorized delegate can claim on their behalf using `delegateClaim(...)`:

```
function claim(uint256[] calldata actualIds, address[] calldata claimTos, uint256 batchId, bytes calldata extraData)
↪  external virtual;

function delegateClaim(uint256[] calldata actualIds, uint256 batchId, bytes calldata extraData) external virtual;
```

To claim tokens via `claim(...)`, the caller must be the owner of the corresponding `TTFutureTokenV2` NFT with the same ID as the `Actual`. Upon a successful claim, the user receives the allocated tokens, and the unlocker pays the claim fee to the `FeeCollector`, managed by the TokenTable team. Through `delegateClaim(...)`, a project-defined delegate can claim tokens on behalf of users, with the tokens sent directly to the respective NFT owners.

The platform includes several security controls that allow the project owner to manage the token distribution process. The owner can cancel specific `Actual` instances, choosing whether any unclaimed tokens are burned or returned to the user. This is done using the `cancel(...)` function. Additionally, the project owner can withdraw deposited tokens via the `withdrawDeposit(...)` function.

```
function cancel(
    uint256[] calldata actualIds,
    bool[] calldata shouldWipeClaimableBalance,
    uint256 batchId,
    bytes calldata extraData
) external virtual returns (uint256[] memory pendingAmountClaimables);

function withdrawDeposit(uint256 amount, bytes calldata extraData) external virtual;
```

6.2 Fee Collector

The `FeeCollector` contract is owned by the **TokenTable** team. Upon each claim, the unlocker contract pays a fee to the `FeeCollector`. The fee amount is determined by the `getFee(...)` function:

```
function getFee(address unlockerAddress, uint256 tokenTransferred) external view returns (uint256 tokensCollected);
```

This function returns the fee based on the unlocker's address and the amount of tokens being transferred to the user. The `FeeCollector` supports two fee models: a fixed fee and a percentage-based fee. The fixed fee is an absolute value, adjusted for the token's decimals, while the percentage fee charges a specified percentage of the claimed amount.

The `FeeCollector` contract is utilized across the entire **TokenTable** platform. Other contracts in the token distribution suite can also use functions such as `getFeeToken`, which returns the token type in which the fee should be paid. By default, fees are collected in the project's native tokens.

7 Issues

7.1 [Medium] Fixed fees allow users to transfer all the project tokens from the `Unlocker` to the protocol owner

File(s): `TokenTableUnlockerV2.sol`

Description: In the `TTUFeeCollector` contract, protocol can set fixed fees for selected unlockers. These fees will get charged and transferred every time users call the `claim(...)` function. However, there is no minimum claim amount required for this function to complete. Users can call this function indefinitely until all the `ProjectTokens` in the `Unlocker` get transferred to the `TTUFeeCollector.owner()` as fees.

Impact: All the project tokens that have been deposited by the team to cover the users' claims will be transferred to the `Token Table` protocol as fees.

Recommendation(s): For fixed fees there should be a reasonable minimum claiming amount.

Status: Fixed

Update from TokenTable: Fixed in `b119c645215fec35ae08aa2c431433ca52a3dce6`

Update from CODESPECT: The scenario involving a zero amount has been taken into account; however, in certain situations, the Fixed Fee may still exceed the amount being claimed.

Update from TokenTable: Acknowledged.

7.2 [Low] Tracker token's `balanceOf` will revert if address owns cancelled actuals

File(s): `TTTrackerTokenV2`

Description: Tracker token is supposed to track an `Unlcoker`'s claimable project tokens. Specifically, the `balanceOf(...)` function should display the amount of currently claimable tokens of the given address. However, the call will always revert if the address owns a cancelled actual. This happens because the `cancel(...)` function `deletes` the `actuals[]` mapping of the actual.

```
function cancel(
    uint256[] calldata actualIds,
    bool[] calldata shouldWipeClaimableBalance,
    uint256 batchId,
    bytes calldata
) external virtual override onlyOwner returns (uint256[] memory pendingAmountClaimables) {
        ..
        delete $.actuals[actualId];
    }
    _callHook(_msgData());
}
```

This mapping is later used by the `calculateAmountClaimable(...)` function which `Tracker Token`'s `balanceOf(...)` function calls.

```
function calculateAmountClaimable(uint256 actualId)
    public
    view
    virtual
    override
    returns (uint256 deltaAmountClaimable, uint256 updatedAmountClaimed)
{
    (deltaAmountClaimable, updatedAmountClaimed) = simulateAmountClaimable(actualId, block.timestamp);
}

function simulateAmountClaimable(uint256 actualId, uint256 claimTimestampAbsolute)
    public
    view
    virtual
    override
    returns (uint256 deltaAmountClaimable, uint256 updatedAmountClaimed)
{
    TokenTableUnlockerV2Storage storage $ = _getTokenTableUnlockerV2Storage();
    Actual memory actual = $.actuals[actualId];
    if (actual.presetId == 0) revert ActualDoesNotExist();
    ..
}
```

Impact: If a user owns a cancelled actual, calling `balanceOf(...)` on that address will always revert.

Recommendation(s): Consider making the `simulateAmountClaimable(...)` function `return 0` if the actual doesn't exist instead of reverting.

Status: Fixed

Update from TokenTable: 4652bc40f9300ebd99a5fc2c26ff20a8cc94fb69

7.3 [Low] Tracker token's `totalSupply` is always incorrect

File(s): `TTTrackerTokenV2`

Description: Tracker token is supposed to track an `Unlcoker`'s claimable project tokens. Specifically, the `totalSupply()` function should track the total number of tokens deposited into the `Unlocker` awaiting claim.

```
/**
 * @dev Total number of tokens deposited into the unlocker awaiting claim.
 */
function totalSupply() external view returns (uint256) {
    return IERC20Metadata(ttuInstance.getProjectToken()).balanceOf(address(this));
}
```

However, the function is returning the `balanceOf(address(this))`, instead of the balance of the `Unlocker` which holds the project tokens.

Impact: Tracker token will return the wrong number of tokens that are deposited in the `Unlocker` as it is tracking the wrong address' balance.

Recommendation(s): Make the function return the `balanceOf(address(ttuInstance))`.

Status: Fixed

Update from TokenTable: 0bff495890939f24db93a77b5c03979419b0f2ad

7.4 [Low] Unauthorized claims allowed when `externalDelegateRegistry` is not configured

File(s): `TokenTableUnlockerV2.sol`

Description: The `TokenTableUnlockerV2` contract allows designated users to claim tokens on behalf of others via the `delegateClaim(...)` function. A caller can perform this action only if they are:

1. Included in the internal `claimingDelegates` set, **or**;
2. Authorised via an external delegate registry (if one is configured);

Only the contract owner can manage the `claimingDelegates` list. If an external registry is configured, it will be used to verify whether the `msg.sender` has permission to make a claim on behalf of the original token owner.

```
function delegateClaim(uint256[] calldata actualIds, ...)
  ...
{
    TokenTableUnlockerV2Storage storage $ = _getTokenTableUnlockerV2Storage();
    bool callerIsClaimingDelegate = $.claimingDelegates.contains(_msgSender());
    for (uint256 i = 0; i < actualIds.length; i++) {
        if (
            // @audit statement incorrect
            !callerIsClaimingDelegate && $.currentChainSupportsExternalDelegateRegistry
                && !externalDelegateRegistry.checkDelegateForContract(
                    _msgSender(), $.futureToken.ownerOf(actualIds[i]), address(this), this.delegateClaim.selector
                )
        ) {
            revert NotPermissioned();
        }
        _claim(actualIds[i], address(0), batchId);
    }
    _callHook(_msgData());
}
```

The logic inside the `if` condition is flawed. The intention is to *revert* if the caller is **not** a claiming delegate **and not** authorised via the registry. However, due to the current structure, this is not correctly enforced when the external registry is **not configured**.

Example case where the registry is not configured:

- `callerIsClaimingDelegate = false;`
- `currentChainSupportsExternalDelegateRegistry = false;`

The condition evaluates to:

```
!false && false => true && false => false
```

This incorrectly allows execution to continue, letting unauthorised users claim on behalf of others if the registry is not enabled.

Impact: This breaks the core invariant of the `delegateClaim` logic. Unauthorised users can claim tokens for others when the external registry is not set, potentially compromising any contracts that integrate with `TokenTableUnlockerV2`.

Recommendation(s): Refactor the conditional logic to clearly enforce that **only** a permitted claiming delegate or a registry-authorized user can claim on behalf of others.

Status: Fixed

Update from TokenTable: 661c34353311d27ad03f35f9066a7a772d5be6d0

7.5 [Low] `defaultFee` can be huge or insignificant depending on the project token's decimals

File(s): `TTUFeeCollector.sol`

Description: Projects can deploy an `Unlocker` for their project token in a permissionless manner through the deployer. If there is no communication or intervention from the protocol team to set up custom fees for the `Unlocker`, then the `defaultFee` is charged every time users call the `claim(...)` function. However, this `defaultFee` is a fixed fee amount and not a percentage.

Projects can have tokens with varying decimals, ranging from as low as 2 to as high as 24. It is impossible for a `defaultFee` to exist that can fairly cover all these decimals.

Impact: Depending on the project token decimals, the `defaultFee` can be huge or very small, requiring the protocol to step in and set up a custom fee. This removes the permissionless nature of the process for the projects.

Recommendation(s): Instead of having the `defaultFee` as a fixed amount, make it a percentage in BIPS.

Status: Acknowledged

Update from TokenTable: acknowledged

7.6 [Info] Deploying an `Unlocker` with Multicall may revert for 0 deposit amounts

File(s): `TTUMulticallDeployer.sol`

Description: The `TTUMulticallDeployer` contracts allows projects to deploy an `Unlocker`, create `presets`, create `actuals` and deposit their project token in one call. A project may not want to deposit tokens into the `Unlocker` yet and desire to do it at a later stage (possibly after custom fees have been set up). However, if the project token reverts at `0` amount transfers, then the multicall will also revert:

```
function multicallDeploy(
    ITTUDeployer deployer,
    bytes memory encodedDeployerParams,
    bytes calldata encodedCreatePresetsParams,
    bytes calldata encodedCreateActualsParams
) external {
    ITokenTableUnlockerV2 unlocker;
    {
        (
            address projectToken,
            address existingFutureToken,
            string memory projectId,
            bool isUpgradeable,
            bool isTransferable,
            bool isCancelable,
            bool isHookable,
            bool isWithdrawable,
            uint256 depositAmount
        ) = abi.decode(encodedDeployerParams, (address, address, string, bool, bool, bool, bool, bool, uint256));
        (unlocker,) = deployer.deployTTSuite(
            projectToken,
            existingFutureToken,
            projectId,
            isUpgradeable,
            isTransferable,
            isCancelable,
            isHookable,
            isWithdrawable
        );

        IERC20(projectToken).transferFrom(msg.sender, address(unlocker), depositAmount);
    }
    ...
}
```

Impact: Deploying with multicall will revert for `0` deposit amounts if the project token reverts for such transfers.

Recommendation(s): Implement a check that only call `transferFrom(...)` if the `depositAmount` is greater than `0`.

Status: Acknowledged

Update from TokenTable: acknowledged

7.7 [Info] Future token's `getClaimInfo` function can return inaccurate information about token's cancel status

File(s): `TTFutureTokenV2.sol`

Description: The `getClaimInfo(...)` function is supposed to return 3 different variables for the `tokenId` provided:

1. The amount of tokens claimable as of now;
2. The amount of tokens claimed as of now;
3. The cancellability of the token's unlocker;

However, the cancellability of the unlocker is not enough to determine if a token is cancelled, as the token could get cancelled first and then by calling the `disableCancel(...)` function it will show that it's not cancellable, which is contradictory to the state of the `tokenId`.

Status: Fixed

Update from TokenTable: Replaced this function in `15637996d75331063f50ecbc2a5b660f33268f4b`

7.8 [Info] Implementation of `CustomERC2771Context` based on the vulnerable version of OZ contract

File(s): `CustomERC2771Context.sol`

Description: `ERC2771/*.sol` extensions inherit from `CustomERC2771Context.sol`, which facilitates gasless transaction execution. This abstract contract is based on OpenZeppelin's `ERC2771Context` implementation, version `4.7.0`.

However, this version of the OpenZeppelin contract contains a known issue disclosed in the following security advisory: [ref]. As cited: *"Contracts using ERC2771Context along with a custom trusted forwarder may see _msgSender return address(0) in calls that originate from the forwarder with calldata shorter than 20 bytes."*

To prevent this, the contract should be updated to align with the latest OpenZeppelin implementation, which addresses this issue.

Impact: If a custom trusted forwarder is used and a transaction includes calldata shorter than 20 bytes, `_msgSender()` will return `address(0)`. This can lead to unexpected behaviour.

Recommendation(s): Upgrade `CustomERC2771Context.sol` to reflect the latest secure implementation provided by OpenZeppelin.

Status: Fixed

Update from TokenTable: Fixed in `5faa20f8fe1c7937ff72b00bb3579d39980a792b`

7.9 [Info] Tracker token's `balanceOf` doesn't account for cancelled actuals

File(s): `TTTrackerTokenV2.sol`

Description: Tracker token is supposed to track an `Unlcoker`'s claimable project tokens. Specifically, the `balanceOf(...)` function should display the amount of currently claimable tokens of the given address.

```
/**
 * @dev Number of currently claimable tokens of the given address.
 */
function balanceOf(address account) external view returns (uint256) {
    uint256 amountClaimable;
    ITTFutureTokenV2 nftInstance = ttuInstance.futureToken();
    uint256[] memory tokenIdsOfOwner = nftInstance.tokensOfOwner(account);
    for (uint256 i = 0; i < tokenIdsOfOwner.length; i++) {
        (uint256 deltaAmountClaimable,) = ttuInstance.calculateAmountClaimable(tokenIdsOfOwner[i]);
        amountClaimable += deltaAmountClaimable;
    }
    return amountClaimable;
}
```

However, this function doesn't account for cancelled actuals. A cancelled actual may still have a claimable amount ready to be claimed which is stored in the `pendingAmountClaimableForCancelledActuals` mapping. The `balanceOf(...)` function only returns the claimable amount of active actuals.

Impact: The `balanceOf(...)` will return incorrect claimable amount for addresses that have cancelled actuals with claimable balance.

Recommendation(s): Also call the `pendingAmountClaimableForCancelledActuals(...)` function to retrieve this amount for cancelled actuals.

Status: Fixed

Update from TokenTable: 881e65e4ff8fa35daf4c6d2d04c6a1b2f65fd026

7.10 [Best Practice] Order of calculation in `simulateAmountClaimable` function

File(s): `TokenTableUnlockerV2.sol`

Description: The calculation order in the `simulateAmountClaimable(...)` function can be optimised to reduce precision loss due to integer division. More specifically, the following line:

```
updatedAmountClaimed = (updatedAmountClaimed * actual.totalAmount) / BIPS_PRECISION / TOKEN_PRECISION;
```

Could be changed to this:

```
updatedAmountClaimed = (updatedAmountClaimed * actual.totalAmount) / (BIPS_PRECISION * TOKEN_PRECISION);
```

Status: Fixed

Update from TokenTable: Fixed in `f2155e0b440808928330ec90aa6003a4c630eb9d`

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

UnlockerV2's compilation output:

```
% forge compile
[] Compiling...
[] Compiling 76 files with Solc 0.8.28
[] Solc 0.8.28 finished in 1.07s
Compiler run successful!
```

9.2 Tests Output

UnlockerV2's test output:

```
% forge test
Ran 11 tests for test/TokenTableUnlockerV2.t.sol:TokenTableUnlockerV2Test
[PASS] test_Core_Initialize() (gas: 13837)
[PASS] test_Core_calculateClaimableAmount() (gas: 2698775)
[PASS] test_Core_createActualAndEnforcePermissions() (gas: 952767)
[PASS] test_Core_createPresetAndEnforcePermissions() (gas: 704600)
[PASS] test_Core_disableCreate() (gas: 605165)
[PASS] test_Core_founderCancelablesCancelAndRefund() (gas: 926109)
[PASS] test_Core_investorClaimCorrectAmount() (gas: 1141412)
[PASS] test_Core_withdrawDepositAndEnforcePermissions() (gas: 801858)
[PASS] test_Deployer_deployAndCompleteBeaconUpgrade() (gas: 3954893)
[PASS] test_Deployer_deployAsCloneAndNotUpgrade() (gas: 3746849)
[PASS] test_TrackerToken_reflectCorrectClaimableAmount() (gas: 1176298)
Suite result: ok. 11 passed; 0 failed; 0 skipped; finished in 43.64ms (24.10ms CPU time)

Ran 13 tests for test/TTUFeeCollector.t.sol:TTUFeeCollectorTest
[PASS] testFuzz_getFee(address,uint256,uint16,uint128,uint256) (runs: 256, : 87319, ~: 90623)
[PASS] testFuzz_getFeeToken(address,address,address) (runs: 256, : 63689, ~: 63689)
[PASS] testFuzz_setCustomFeeBips_fail_feesTooHigh(address,uint256) (runs: 256, : 13970, ~: 13970)
[PASS] testFuzz_setCustomFeeBips_fail_notOwner(address,address,uint256) (runs: 256, : 15144, ~: 15144)
[PASS] testFuzz_setCustomFeeBips_success(address,uint256) (runs: 256, : 37137, ~: 38770)
[PASS] testFuzz_setCustomeFeeToken_success(address,address) (runs: 256, : 36320, ~: 36398)
[PASS] testFuzz_setDefaultFeeToken_success(address) (runs: 256, : 35175, ~: 35253)
[PASS] testFuzz_setDefaultFee_fail_notOwner(address,uint256) (runs: 256, : 13417, ~: 13417)
[PASS] testFuzz_setDefaultFee_success(uint256) (runs: 256, : 35408, ~: 35719)
[PASS] testFuzz_withdrawFee_fail_notOwner(address,uint256) (runs: 256, : 73589, ~: 74678)
[PASS] testFuzz_withdrawFee_fail_wrongToken(address,uint256) (runs: 256, : 77458, ~: 77458)
[PASS] testFuzz_withdrawFee_success_erc20(uint256) (runs: 256, : 93030, ~: 93242)
[PASS] testFuzz_withdrawFee_success_ether(uint128) (runs: 256, : 18307, ~: 18360)
Suite result: ok. 13 passed; 0 failed; 0 skipped; finished in 73.45ms (475.37ms CPU time)

Ran 2 test suites in 81.65ms (117.10ms CPU time): 24 tests passed, 0 failed, 0 skipped (24 total tests)
```

9.3 Notes about Test suite

The TokenTable team delivered a robust and comprehensive test suite for most functionalities, but lacked tests for a few functions like `delegateClaim` where a simple issue was found. Key highlights of the test suite include:

- **Fuzzing Tests:** The extensive use of fuzzing tests enabled coverage of numerous edge cases, particularly for the fee collection contract. These tests validate the behaviour of the system under a wide range of inputs, uncovering potential vulnerabilities or inconsistencies that could emerge in unexpected conditions.
- **Complex Scenarios:** Beyond basic operations, the test suite included sophisticated scenarios such as creating "Actuals" with an already random claimed amount. These tests verified the correctness and stability of the system in handling advanced operational flows, reinforcing confidence in the protocol's ability to operate effectively under dynamic conditions.

While the suite's depth is commendable, ongoing improvements, such as extending fuzzing ranges and incorporating even more complex operational scenarios for every functionality of the procotol, could further solidify its effectiveness.