Eco Permit3 PR 28 & 37

Security Review

Cantina Managed review by:

Rikard Hjort, Lead Security Researcher
0xhuy0512, Associate Security Researcher

September 16, 2025

Contents

1 Introduction **2**
1.1 About Cantina . . . 2
1.2 Disclaimer . . . 2
1.3 Risk assessment . . . 2
1.3.1 Severity Classification . . . 2

2 Security Review Summary **3**

3 Permit 3 PR 28 Findings **4**
3.1 High Risk . . . 4
3.1.1 Signature validation incorrectly handles contract and short signatures . . . 4
3.1.2 Transfer permits bypass lockdown functionality . . . 4
3.2 Medium Risk . . . 5
3.2.1 Allowance timestamp can not be decreased . . . 5
3.2.2 Anyone can issue approvals for ERC7702 delegatee . . . 5
3.3 Low Risk . . . 6
3.3.1 No way to reset an existing allowance . . . 6
3.4 Gas Optimization . . . 6
3.4.1 Redundant zero address check for owner . . . 6
3.4.2 Unnecessary checks on address parameters . . . 7
3.4.3 Redundant check for empty salts array . . . 7
3.4.4 `NONCE_` constants can be `bool`s . . . 7
3.5 Informational . . . 8
3.5.1 Lockdown issues . . . 8

4 Permit 3 PR 37 Findings **9**
4.1 High Risk . . . 9
4.1.1 Collection-wide lockdown can be bypassed by specific token approvals . . . 9
4.2 Medium Risk . . . 9
4.2.1 Transferring with Permit3 contract are not possible for ERC721/ERC1155 with `tokenId` of `uint256.max` . . . 9
4.3 Informational . . . 10
4.3.1 Overloaded function names are confusing . . . 10
4.3.2 Incorrect NatSpec comments . . . 11
4.3.3 Approval event for NFTs lacks `tokenId` . . . 11
4.3.4 Ambiguous `Transfer` name in `PermitType` enum . . . 11
4.3.5 Non-valid addresses allowed for Transfers . . . 12
4.3.6 The struct `MultiTokenTransfer` is misleadingly named . . . 12
4.3.7 Inefficiency on batch ERC1155 transfers . . . 12

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings are a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Eco enables apps to unlock stablecoin liquidity from any connected chain and give users the simplest onchain experience.

From Aug 23rd to Aug 29th the Cantina team conducted a review of permit3 on commit hash 833b84f8 (PR 28). The team identified a total of **10** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	2	1	1
Medium Risk	2	1	1
Low Risk	1	1	0
Gas Optimizations	4	4	0
Informational	1	0	1
Total	**10**	**7**	**3**

Next, from Sep 9th to Sep 16th the Cantina team conducted a review of permit3 on commit hash 70649332 (PR 37). The team identified a total of **9** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	1	1	0
Medium Risk	1	1	0
Low Risk	0	0	0
Gas Optimizations	0	0	0
Informational	7	7	0
Total	**9**	**9**	**0**

3 Permit 3 PR 28 Findings

3.1 High Risk

3.1.1 Signature validation incorrectly handles contract and short signatures

Severity: High Risk

Context: NonceManager.sol#L216-L233

Description: The `_verifySignature()` function in `NonceManager` contract attempts to support standard 65-byte signatures, ERC-2098 64-byte signatures, and ERC-1271 contract signatures. However, the implementation has two critical flaws.

First, when `signature.length` is 64, the code calls `digest.recover(signature)`. This will always fail because the underlying `ECDSA.recover(bytes32, bytes)` from OpenZeppelin rejects signatures that are not exactly 65 bytes long, as stated in the library's documentation (ECDSA.sol#L113). This completely prevents the validation of valid 64-byte signatures.

Second, if the `owner` is a smart contract that produces a 64 or 65-byte signature, the logic first attempts validation via `digest.recover(signature)`. This EOA-specific recovery will likely revert when used with a contract signature, as described (ECDSA.sol#L190-L192). Because this check happens before the ERC-1271 check, the contract signature is never validated, effectively locking out smart contract signatures that use 64 or 65-byte signature schemes.

Recommendation: The signature validation logic should first determine if the `owner` is an EOA or a contract. Based on that, it should apply the correct validation method. For EOAs, it should handle 64-byte and 65-byte signatures distinctly. For contracts, it should proceed directly to ERC-1271 validation. The following diff implements the recommended fix:

```
if (owner.code.length == 0) {
    if (signatureLength == 65) {
        if (digest.recover(signature) != owner) {
            revert InvalidSignature(owner);
        }
    } else if (signatureLength == 64) {
        (bytes32 r, bytes32 vs) = abi.decode(signature, (bytes32, bytes32));
        if (digest.recover(r, vs) != owner) {
            revert InvalidSignature(owner);
        }
    } else {
        revert InvalidSignature(owner);
    }
} else {
    if (!owner.isValidERC1271SignatureNow(digest, signature)) {
        revert InvalidSignature(owner);
    }
}
```

Eco Foundation: Fixed in PR 40.

Cantina Managed: Fix verified.

3.1.2 Transfer permits bypass lockdown functionality

Severity: High Risk

Context: Permit3.sol#L347

Description: `lockdown()` is a mechanism for stopping transfers to protect from theft after an account or contract compromise, or a an attacker obtaining a signature. However, the check is bypassed for transfer permits, which can still perform transfers that would otherwise be blocked.

Proof of Concept: Add this test to `Permit3.t.sol`:

```
function test_transferFromLockedAllowance() public {
    uint256 ownerBal = token.balanceOf(owner);

    // Setup approval
    vm.prank(owner);
    permit3.approve(address(token), spender, AMOUNT, uint48(block.timestamp + 1 hours));
    // Sign a permit to an attacker
    IPermit3.ChainPermits memory chainPermits = Permit3TestUtils.createTransferPermit(address(token), spender,
    ↪  AMOUNT);
    uint48 deadline = uint48(block.timestamp + 2 hours);
    uint48 timestamp = uint48(block.timestamp);
    bytes memory signature = _signPermit(chainPermits, deadline, timestamp, SALT);

    // Lock the allowance
    IPermit.TokenSpenderPair[] memory pairs = new IPermit.TokenSpenderPair[](1);
    pairs[0] = IPermit.TokenSpenderPair({ token: address(token), spender: spender });

    vm.prank(owner);
    permit3.lockdown(pairs);

    // Attempt transfer should fail due to locked allowance
    vm.prank(spender);
    vm.expectRevert(abi.encodeWithSelector(IPermit.AllowanceLocked.selector, owner, address(token), spender));
    permit3.transferFrom(owner, recipient, AMOUNT, address(token));

    // Verify allowance is still locked
    (uint160 amount, uint48 expiration,) = permit3.allowance(owner, address(token), spender);
    assertEq(amount, 0);
    assertEq(expiration, 2); // LOCKED_ALLOWANCE = 2
    assertEq(token.balanceOf(owner), ownerBal);

    // Transfer executes even under lockdown
    permit3.permit(owner, SALT, deadline, timestamp, chainPermits.permits, signature);
    assertEq(token.balanceOf(owner), ownerBal);
}
```

Recommendation: Call `_validateLockStatus()` before transferring. Alternatively, add some other mechanism to lock down transfers.

Eco Foundation: Acknowledged.

Cantina Managed: Acknowledged.

3.2 Medium Risk

3.2.1 Allowance timestamp can not be decreased

Severity: Medium Risk

Context: Permit3.sol#L496-L501

Description: There is no way to decrease an allowance timestamp. Thus, if it is ever set too high (e.g. to `type(uint48).max`) then it cannot be decreased. It is essentially treated as a growing nonce with no restriction on its upper bound, when it is supposed to be treateda as the point in time of signing.

Recommendation: Check that `block.timestamp >= timestamp` to prevent too large timestamps to be misused.

Eco Foundation: Fixed in PR 35.

Cantina Managed: Fix verified.

3.2.2 Anyone can issue approvals for ERC7702 delegatee

Severity: Medium Risk

Context: ERC7702TokenApprover.sol#L41-L56

Description: EIP 7702 introduces the possibility of setting code for an EOA. This code *persists*.

The `ERC7702TokenApprover` module has a function, `approve(address[])`, which allows approving multiple tokens on the deployed `Permit3` contract. The NatSpec comment on the `approve()` function states:

```
The EOA must have authorized delegation to this contract in the same transaction
```

This is not correct. The delegation will persist. It is possible, after delegation, for anyone to call `approve()` with any data, on behalf of the delegatee. The result is that the delegatee may approve, for the `Permit3` contract, tokens they did not intend to approve.

Proof of Concept: In `ERC7702ApproverTest`:

```
function test_Approve_OtherUser() public {
    // EXISTING TEST:

    // The owner sets a delegation on their EOA and sends an approval.

    address[] memory tokens = new address[](1);
    tokens[0] = address(token1);

    // Use proper EIP-7702 cheatcodes
    Vm.SignedDelegation memory signedDelegation = vm.signDelegation(address(approver), ownerPrivateKey);

    vm.startPrank(owner);
    vm.attachDelegation(signedDelegation);
    ERC7702TokenApprover(owner).approve(tokens);
    vm.stopPrank();

    assertEq(token1.allowance(owner, address(permit3)), type(uint256).max);

    // NEW PART:

    // Now Bob creates more approvals.

    address[] memory more_tokens = new address[](1);
    IERC20 usdc = IERC20(address(new MockERC20("Circle USD", "USDC")));
    more_tokens[0] = address(usdc);
    address bob = makeAddr("BOB");
    vm.startPrank(bob);
    ERC7702TokenApprover(owner).approve(more_tokens);
    vm.stopPrank();

    assertEq(usdc.allowance(owner, address(permit3)), type(uint256).max);
}
```

Recommendation: Document the behavior prominently and only use the delegation contract for accounts for which the above behavior is appropriate.

Eco Foundation: Acknowledged. This is by design. The idea behind `ERC7702TokenApprover` is to give Permit3 access to all tokens without user needing to sign approval or send transactions for individual tokens.

Cantina Managed: Acknowledged.

3.3 Low Risk

3.3.1 No way to reset an existing allowance

Severity: Low Risk

Context: PermitBase.sol#L70-L71

Description: `approve()` disallows resetting an allowance to 0. This means an existing allowance can't be removed this way. Setting an `approve()` to `0` is the standard way to remove allowances in most tokens and a sensible pattern.

Recommendation: Allow passing a `0` amount to `approve()`.

Eco Foundation: Fixed in PR 34.

Cantina Managed: Fix verified.

3.4 Gas Optimization

3.4.1 Redundant zero address check for owner

Severity: Gas Optimization

Context: Permit3.sol#L109-L111, Permit3.sol#L160-L162, Permit3.sol#L218-L220, Permit3.sol#L280-L282

Description: In `Permit3` contract, functions such as `permit()`, `permitTransferFrom()`, `permitWitnessTransferFrom()`, and `permit2()`, there is a check to ensure the `owner` is not the zero address which is redundant. The `_verifySignature()` function, which is called in these flows, internally relies on `ECDSA.recover()`. The `ECDSA.recover()` function will treat `address(0)` signer as invalid signature and will revert right after Therefore, performing the same check again in the calling functions is unnecessary.

Recommendation: Remove the explicit zero-address check for the `owner` in the affected functions within `Permit3` contract. The `_verifySignature()` function sufficiently handles this validation.

Eco Foundation: Fixed in PR 31.

Cantina Managed: Fix verified.

3.4.2 Unnecessary checks on address parameters

Severity: Gas Optimization

Context: PermitBase.sol#L92-L100

Description: In `PermitBase.sol`, the `transferFrom()` function performs checks to ensure that the `from`, `to`, and `token` addresses are not `address(0)`. However, these checks are redundant as the subsequent call to `_transferFrom()` already includes these validations.

Recommendation: Remove the redundant zero-address checks from the `transferFrom()` function to optimize gas usage.

Eco Foundation: Fixed in PR 32.

Cantina Managed: Fix verified.

3.4.3 Redundant check for empty salts array

Severity: Gas Optimization

Context: NonceManager.sol#L103-L105

Description: In the `invalidateNonces()` function at `NonceManager` contract, there is a check to ensure the `salts` array is not empty. However, this check is redundant because the subsequent call to `_processNonceInvalidation()` will revert if the array is empty.

Recommendation: Remove the explicit check for `salts.length == 0` in `invalidateNonces()` function.

Eco Foundation: Fixed in PR 33.

Cantina Managed: Fix verified.

3.4.4 `NONCE_` constants can be `bool`s

Severity: Gas Optimization

Context: NonceManager.sol#L25-L28

Description: Instead of using the magic numbers `0` and `1` to represent if a nonce has been used, they can be the booleans `false` and `true` respectively. The semantics remain unchanged but then there is no need for the `== NONCE_NOT_USED` comparisons, and the values in the mapping can instead be used directly.

Recommendation: Set `NONCE_NO_USED` to `false`, `NONCE_USED` to `true`, and remove the comparisons from the usage sites:

```diff
- if (usedNonces[owner][salt] != NONCE_NOT_USED) {
+ if (usedNonces[owner][salt]) {
```

```diff
- return usedNonces[owner][salt] == NONCE_USED;
+ return usedNonces[owner][salt];
```

Eco Foundation: Fixed in PR 36.

Cantina Managed: Fix verified.

3.5 Informational

3.5.1 Lockdown issues

Severity: Informational

Context: PermitBase.sol#L159-L183

Description: In case of a compromise of token logic, there is no way to lock down all allowances of a token. (Workaround, revoke the allowance to the `Permit3` contract on that token.).

Recommendation: Allow locking down the entire account of an address, blocking all transfers of any token to any address. This helps with the common problem of revoking many permissions at once and makes it easy for users to apply some hygiene to their existing approvals.

Eco Foundation: Acknowledged.

Cantina Managed: Acknowledged.

4 Permit 3 PR 37 Findings

4.1 High Risk

4.1.1 Collection-wide lockdown can be bypassed by specific token approvals

Severity: High Risk

Context: MultiTokenPermit.sol#L93-L96, MultiTokenPermit.sol#L128-L131, PermitBase.sol#L140-L165

Description: The `lockdown()` function in `PermitBase` contract is designed as an emergency stop, allowing a token owner to revoke all permissions for a spender on a given token contract by setting a collection-wide lock. This lock is signified by setting the `expiration` field of the allowance to `LOCKED_ALLOWANCE`.

The ERC721 and ERC1155 `transferFrom()` functions in `MultiTokenPermit` contract implement a dual-allowance system that first checks for a specific `tokenId` approval. If that fails, it falls back to checking for a collection-wide approval.

The root cause of the vulnerability is that these `transferFrom()` functions check for the specific `tokenId` approval *before* verifying if a collection-wide lock is in place. As a result, if a spender has an approval for a specific token, the transfer will succeed, completely bypassing the collection-wide `lockdown()`.

This flaw defeats the purpose of the emergency `lockdown()` function. An owner who locks a collection to prevent a compromised spender from draining assets will be unsuccessful if that spender holds any specific `tokenId` approvals, which would allow the spender to proceed with transferring those specific tokens.

Recommendation: To enforce the lockdown mechanism correctly, a check for the collection-wide lock must be performed at the beginning of the `transferFrom()` functions for both ERC721 and ERC1155 tokens, before any other allowance logic is executed. Apply the following changes to `MultiTokenPermit` contract:

```
 function transferFrom(address from, address to, address token, uint256 tokenId) public override {
+     // Explicitly check for a collection-wide lock before proceeding.
+     bytes32 collectionKeyCheck = bytes32(uint256(uint160(token)));
+     if (allowances[from][collectionKeyCheck][msg.sender].expiration == LOCKED_ALLOWANCE) {
+         revert AllowanceLocked(from, collectionKeyCheck, msg.sender);
+     }
+
      // Get the encoded identifier for this specific token ID
      bytes32 encodedId = _getTokenKey(token, tokenId);

      // ...
```

Eco Foundation: Fixed in PR 40.

Cantina Managed: Fix verified.

4.2 Medium Risk

4.2.1 Transferring with Permit3 contract are not possible for ERC721/ERC1155 with `tokenId` of `uint256.max`

Severity: Medium Risk

Context: MultiTokenPermit.sol#L23-L31, MultiTokenPermit.sol#L104-L108, MultiTokenPermit.sol#L140-L143

Description: The `_getTokenKey()` function in `MultiTokenPermit` contract uses `type(uint256).max` as a reserved `tokenId` to signify a collection-wide approval. When this `tokenId` is provided, the function returns a key based only on the token address, `bytes32(uint256(uint160(token)))`, which is identical to the key used for collection-wide approvals.

The root cause is the incorrect assumption that `type(uint256).max` is not a valid `tokenId` for an ERC721/ERC1155. The ERC721/ERC1155 standard does not restrict the range of `tokenIds`. Some projects utilize the full `uint256` range for token IDs. For instance, the `Wrapped Number Board` collection includes a token with a `tokenId` of `type(uint256).max`, as documented in research on large or custom ERC721 token IDs

As a result, it is impossible to transfer an ERC721/ERC1155 token where `tokenId == type(uint256).max` with Permit3 contract.

Recommendation: To resolve this collision, the special case for `type(uint256).max` tokenId should be removed from `_getTokenKey()`. This function should consistently hash the `token` address and `tokenId` to generate a unique key for specific token approvals.

Collection-wide approvals should be handled separately by calling the `approve(address,address,uint160,uint48)` function from `PermitBase`. The documentation should be updated to clarify this distinction.

Modify `_getTokenKey()` in `MultiTokenPermit` contract as follows:

```
- if (tokenId == type(uint256).max) {
-     // ERC20 or collection-wide approval - convert address to bytes32
-     return bytes32(uint256(uint160(token)));
- } else {
-     // Specific token ID - hash token and tokenId together
-     return keccak256(abi.encodePacked(token, tokenId));
- }
+ // Specific token ID - hash token and tokenId together
+ return keccak256(abi.encodePacked(token, tokenId));
```

And modify `transferFrom()` as follows:

```
  function transferFrom(address from, address to, address token, uint256 tokenId) public override {
      // ...

      if (revertDataPerId.length > 0) {
          bytes32 collectionKey = bytes32(uint256(uint160(token)));

-         if (encodedId == collectionKey) {
-             _revert(revertDataPerId);
-         }

          (, bytes memory revertDataWildcard) = _updateAllowance(from, collectionKey, msg.sender, 1);

          _handleAllowanceError(revertDataPerId, revertDataWildcard);
      }
      // ...
  }
```

Eco Foundation: Fixed in PR 40.

Cantina Managed: Fix verified.

4.3 Informational

4.3.1 Overloaded function names are confusing

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: In `MultiTokenPermit` contract, the function names `transferFrom()` and `batchTransferFrom()` are heavily overloaded to handle transfers for different token standards (ERC721, ERC1155) and various batching strategies. While this is valid Solidity, it reduces code clarity. Discerning a function's precise behavior requires inspecting the specific arguments passed, making the contract's interface less intuitive and harder to integrate with correctly. More explicit function names would improve readability and reduce the chance of developer error.

Recommendation: For better code clarity and developer experience, consider renaming the overloaded functions to reflect their specific purpose.

```
- transferFrom(address,address,address,uint256)
+ transferERC721From(address,address,address,uint256)


- transferFrom(address,address,address,uint256,uint160)
+ transferERC1155From(address,address,address,uint256,uint160)
```

```diff
- transferFrom(ERC721TransferDetails[] calldata)
+ batchTransferERC721From(ERC721TransferDetails[] calldata)


- transferFrom(MultiTokenTransfer[] calldata)
+ batchTransferERC1155From(MultiTokenTransfer[] calldata)


- batchTransferFrom(ERC1155BatchTransferDetails calldata)
+ batchTransferERC1155From(ERC1155BatchTransferDetails calldata)


- batchTransferFrom(TokenTypeTransfer[] calldata)
+ batchTransferMultiTokenTypeFrom(TokenTypeTransfer[] calldata)
```

Eco Foundation: Fixed in PR 40.

Cantina Managed: Fix verified.

4.3.2 Incorrect NatSpec comments

Severity: Informational

Context: MultiTokenPermit.sol#L20, MultiTokenPermit.sol#L38, MultiTokenPermit.sol#L59

Description: The NatSpec documentation for `_getTokenKey()`, `allowance()`, and `approve()` in `MultiTokenPermit` contract incorrectly states that a `tokenId` of `0` is used to refer to ERC20 tokens. This is misleading, as it is not a special value reserved for ERC20s.

Recommendation: Update the NatSpec comments to remove the incorrect reference to `tokenId = 0` for ERC20s and clarify the mechanism for collection-wide approvals.

Eco Foundation: Fixed in PR 40.

Cantina Managed: Fix verified.

4.3.3 Approval event for NFTs lacks `tokenId`

Severity: Informational

Context: MultiTokenPermit.sol#L80

Description: The `approve()` function in `MultiTokenPermit` contract is designed to handle approvals for specific ERC721 and ERC1155 tokens by accepting a `tokenId`. However, it only emits the standard `Approval` event, which omits the `tokenId`. This makes it difficult for off-chain services to accurately track which specific token has been approved. Without the `tokenId` in the event, monitoring systems can only see that an approval has been made on the contract level, not for the individual asset, reducing transparency.

Recommendation: To provide full visibility for off-chain services, a new event should be introduced that includes the `tokenId`. The `approve()` function should then emit this new event when a specific token is being approved, rather than the generic `Approval` event.

Eco Foundation: Fixed in PR 40.

Cantina Managed: Fix verified.

4.3.4 Ambiguous `Transfer` name in `PermitType` enum

Severity: Informational

Context: IPermit3.sol#L20

Description: The `PermitType` enum in `IPermit3` interface includes a member named `Transfer`. Within a contract that now supports multiple token standards (ERC20, ERC721, and ERC1155), this generic name is ambiguous. The underlying logic that processes this permit type exclusively handles ERC20 transfers by calling the `_transferFrom()` function inherited from `PermitBase` contract. This ambiguity could mislead into incorrectly assuming this `Transfer` type can be used for any token standard, leading to integration errors and reverted transactions when attempting to use it for NFTs.

Recommendation: To improve clarity and prevent potential misuse, rename the `Transfer` enum member to `TransferERC20` to explicitly signify its intended functionality.

Eco Foundation: Fixed in PR 40.

Cantina Managed: Fix verified.

4.3.5 Non-valid addresses allowed for Transfers

Severity: Informational

Context: Permit3.sol#L338

Description: A `PermitType.Transfer` operation should always have a token address attached. The current implementation treats any `bytes32` value as a valid address by truncating it to 20 bytes. This provides no benefit and can make transfers more opaque.

Recommendation: Use `AddressConverter` and `address token = p.tokenKey.toAddress();` to prevent any incorrect use of non-address bytes32 values. Alternatively, check the top 12 bytes manually:

```
if ((p.tokenKey >> 160) != 0) {
    revert InvalidTransferAddress()
}
```

Eco Foundation: Fixed in PR 40.

Cantina Managed: Fix verified.

4.3.6 The struct `MultiTokenTransfer` is misleadingly named

Severity: Informational

Context: MultiTokenPermit.sol#L177-L179

Description: The `transferFrom()` function in `MultiTokenPermit` contract processes batch ERC1155 transfers. However, it accepts an array of `MultiTokenTransfer` structs, a name that incorrectly implies support for multiple token types.

Recommendation: Rename the `MultiTokenTransfer` struct to something more specific that accurately reflects its exclusive use for ERC1155 tokens in this context, such as `ERC1155TransferDetails`.

Eco Foundation: Fixed in PR 40.

Cantina Managed: Fix verified.

4.3.7 Inefficiency on batch ERC1155 transfers

Severity: Informational

Context: MultiTokenPermit.sol#L207-L211

Description: The `batchTransferFrom()` function processes multiple ERC1155 transfers by iterating through each token and calling the single `ERC1155.safetransferFrom()` function internally.

Recommendation: First, update all necessary allowances within a loop. After the loop completes, perform a single, aggregated `ERC1155.safeBatchTransferFrom()` call with all the token IDs and amounts. This separates the allowance checks from the external transfer call, improving efficiency.

Eco Foundation: Fixed in PR 40.

Cantina Managed: Fix verified.