CANTINA

Settlement contracts

Security Review

Cantina Managed review by:

0xWeiss, Security Researcher
Víctor Martínez, Security Researcher

April 20, 2025

Contents

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	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

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 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

t3rn is a Modular Interoperability Layer designed for fast, secure, and cost-efficient cross-chain swapping.

From Feb 28th to Mar 8th the Cantina team conducted a review of settlement-contracts on commit hash 4fca4324. The team identified a total of **19** issues:

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	0	0	0
High Risk	2	2	0
Medium Risk	5	4	1
Low Risk	11	11	0
Gas Optimizations	0	0	0
Informational	1	1	0
Total	**19**	**18**	**1**

3 Findings

3.1 High Risk

3.1.1 DoS on refunds

Severity: High Risk

Context: avpBatchSubmitter.sol#L286, remoteOrder.sol#L571

Description: When calling `claimRefundV2` if there is a reward claimable for certain beneficiary, it will enter the following if statement:

```
if (claimer.isClaimable(_batchPayloadHash, _batchPayload, orderId, _beneficiary, maxReward, 1)) {
    escrowGMP.twoifyPayloadHash(orderId);
    ro.settlePayoutWithFeesCall(maxReward, rewardAsset, _beneficiary, address(ro), 1, orderId);
} else {
    emit NonRefundable(orderId, escrowGMP.getRemotePaymentPayloadHash(orderId), _batchPayloadHash);
}
```

After, if forwards the call to the `settlePayoutWithFeesCall` function inside the remote order contract where it tries to send funds via `settleNativeOrToken`:

```
function settleNativeOrToken(
    uint256 amount,
    address asset,
    address beneficiary,
    address sender
) internal nonReentrant returns (bool) {
    if (amount == 0) return true;
    if (beneficiary == address(0)) return false;
    if (asset == address(0)) {
        (bool sent, ) = beneficiary.call{value: amount}("");
        return sent;
    }

    IERC20(asset).safeTransferFrom(sender, beneficiary, amount);
    return true;
}
```

Notice both of the key parameters, `sender` and `beneficiary`. According to the first call in the `claimRefundV2` functions, the `sender` was specified to be the remote order contract `address(ro)`, which in this case it does act like `address(this)`. `safeTransferFrom` requires a pre-approval for the funds to be transferred and when calling it from the same contract it will revert.

Recommendation: If the sender address is meant to be `address(this)` (the remote order contract) use `safeTransfer` instead than `safeTransferFrom`.

t3rn: Fixed in commit 590e0e8e.

Cantina Managed: Fix verified.

3.1.2 Nullify Order ID status after confirmation in `confirmOrderV3`

Severity: High Risk

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

Description: In the `confirmOrderV3` function, the order ID is not nullified after confirmation, which allows the same order to potentially be confirmed by two different relayers if the transactions are executed in the same block. This could lead to a scenario where only one of the two relayers that fulfilled and confirmed the order would be able to claim the payout on the source chain. A hash of `amount`, `asset`, `target`, and `orderId` can be used as a status key to ensure a unique mapping for the oder status.

Recommendation: For the status mapping a hash of `amount`, `asset`, `target`, and `orderId` can be used.

1. Add a check at the beginning of the function to verify if the order has already been confirmed by checking the confirmation status using a hash of `amount`, `asset`, `target`, and `orderId`.

2. After successfully confirming the order, mark it as confirmed in storage by setting a true value in the confirmation status mapping (using the hash as the key). This ensures that once an order (and parameters) is confirmed, it cannot be re-confirmed by another relayer, preventing the second relayer from losing the funds.

t3rn: Fixed in commit 4bf7371.

Cantina Managed: Fix verified.

3.2 Medium Risk

3.2.1 Operators can't be removed in case they act maliciously

Severity: Medium Risk

Context: avpBatchSubmitter.sol#L82

Description: Currently, the owner can whitelist certain operators which are granted certain permissions to call multiple functions:

```
function setOperator(address _operator) external onlyOwner {
    operators[_operator] = true;
}
```

This lacks support to remove operators in case they act maliciously or even a mistake has been made when adding them as the boolean is hardcoded to be `true`.

Recommendation: Add the ability to remove operators.

t3rn: Fixed in commit 909b8e3.

Cantina Managed: Fix verified.

3.2.2 Withdrawals can silently fail

Severity: Medium Risk

Context: remoteOrder.sol#L576

Description: When calling `emergencyWithdraw` for a certain `beneficiary` it does make an external call trying to send ETH to the beneficiary specified:

```
function emergencyWithdraw(address asset, uint256 amount, address beneficiary) external onlyOwner {
    settleNativeOrToken(amount, asset, beneficiary, address(this));
```

Inside this call, if there was an unsuccessful transfer, it would return false, which it is not checked for after.

```
if (asset == address(0)) {
    (bool sent, ) = beneficiary.call{value: amount}("");
    return sent;
}
```

Recommendation: Check that the return was not false, if so, revert.

```
  function emergencyWithdraw(address asset, uint256 amount, address beneficiary) external onlyOwner {
-     settleNativeOrToken(amount, asset, beneficiary, address(this));
+     require(settleNativeOrToken(amount, asset, beneficiary, address(this)), "transfer failed");
```

t3rn: Fixed in commit 590e0e8e.

Cantina Managed: Fix verified.

3.2.3 Beneficiaries can grief senders on ETH transfers

Severity: Medium Risk

Context: remoteOrder.sol#L327

Description: Functions such as `confirmOrderV3` allow a sender to the send funds to a beneficiary via a external call in case of the asset being ETH.

```
require(settleNativeOrToken(amount, asset, target, msg.sender), "RO#2");
```

The function makes an external call to the beneficiary where they can basically spend the remaining gas of the transaction and make the sender pay for a huge increase on gas fees:

```
if (asset == address(0)) {
    (bool sent, ) = beneficiary.call{value: amount}("");
    return sent;
}
```

Recommendation: As limiting the gas being forwarded does not prevent a return bomb attack, the only choice is to use a low level call to avoid loading the returned data into memory:

```
assembly {
    success := call(gasLimit, receiver, amount, 0, 0, 0, 0)
}
```

t3rn: Fixed in commit 6bc61577.

Cantina Managed: Fix verified.

3.2.4 Bonuses can't be applied

Severity: Medium Risk

Context: bonusesL3.sol#L8

Description: The `BonusesL3` contract expects to hold ETH in its balance to later distribute it but it lacks a receive function:

```
function distribute(address to, uint256 amount) internal nonReentrant returns (bool) {
    if (to == address(0)) {
        emit DistributionAttempt(to, amount, false, "Invalid address");
        return false;
    }
    if (amount == 0) {
        emit DistributionAttempt(to, amount, false, "Invalid amount");
        return false;
    }
    if (address(this).balance < amount) {
        emit DistributionAttempt(to, amount, false, "Insufficient balance");
        return false;
    }
    (bool success, ) = to.call{value: amount}("");
    emit DistributionAttempt(to, amount, success, success ? "Success" : "Payment failed");
    return success;
}
```

Recommendation: Add a receive function inside the `BonusesL3` contract.

t3rn: Fixed in commit 51d7e72.

Cantina Managed: Fix verified.

3.2.5 Same second orders will fail given a timestamp nonce usage

Severity: Medium Risk

Context: remoteOrder.sol#L197

Description: The nonce for orders is calculated using both the timestamp and the address of the msg.sender:

```
function orderMemoryData(bytes memory input) public payable isOn returns (bytes32) {
    uint32 nonce = uint32(block.timestamp);
    bytes32 id = generateId(msg.sender, nonce);
```

Nonces should not be based on timestamp, they should be their own variable that it is incremented by 1 every time the function gets called. If called twice during the same second, in this case, it would revert inside `storeRemoteOrderPayload`:

```
require(
    escrowGMP.storeRemoteOrderPayload(id, keccak256(abi.encode(rewardAsset, maxReward, block.timestamp))),
    "RO#0"
);
```

Notice how even if it would go through there would be no distinction in the event emission, as no real nonce is used here, and the same event would be emitted:

```
emit RemoteOrderCreated(id, nonce, msg.sender, block.timestamp);
```

Recommendation: A nonce should be used as an incrementable variable each time a user calls that function, not based on timestamp.

t3rn: We've changed nonce to timestamp after recommendation of Quanstamp, it used to be implemented based on increments of nonce per account stored in separate mapping before that. The security isn't breached here IMO since the call would fail on adding the duplicated record to `escrowGMP`.

Cantina Managed: Acknowledged.

3.3 Low Risk

3.3.1 `GMPExpectedPayloadNotMatched` reflects the wrong beneficiary

Severity: Low Risk

Context: escrowGMP.sol#L180

Description: When the payload hash is not found in the `commitEscrowBeneficiaryPayload` function, then the `GMPExpectedPayloadNotMatched` event is emitted to reflect such case:

```
// Update the payment payload (hash of the payload)
bytes32 currentHash = escrowOrdersPayloadHash[sfxId];
if (currentHash == bytes32(0)) {
    emit GMPExpectedPayloadNotMatched(sfxId, address(0), currentHash);
    return (false);
}
```

The address of the beneficiary is hardcoded to 0 while it should be the actual beneficiary passed as a function argument, same as done in the `commitRemoteBeneficiaryPayload` function.

Recommendation: Update the address to reflect the beneficiary instead of `address(0)`:

```
- emit GMPExpectedPayloadNotMatched(sfxId, address(0), currentHash);
+ emit GMPExpectedPayloadNotMatched(sfxId, beneficiary, currentHash);
```

t3rn: Fixed in commit 9212cc2.

Cantina Managed: Fix verified.

3.3.2 Use pull over push method for handling protocol fees

Severity: Low Risk

Context: remoteOrder.sol#L532

Description: Currently, a push system is used when paying the protocol fee as the fee is sent in each transaction to the protocol address. This causes multiple problems:

- For every transaction the gas costs increase significally as it will process the extra logic for sending the fee to the fee address every time.
- There is a DoS scenario where the protocol could block the users from claiming refunds as they could make the call that sends the fee to their address revert.
- Similarly, if the keys get leaked the protocol fee address can be taken over and willingly DoS refunds.

```
function settlePayoutWithFees(uint256 amount, address asset, address beneficiary, address sender) internal {
    uint256 fees = currentProtocolFee > 0 ? calcProtocolFee(amount) : 0;
    require(
        settleNativeOrToken(fees, asset, protocolFeesCollector, sender) &&
            settleNativeOrToken(amount - fees, asset, beneficiary, sender),
        "RO#16"
    );
}
```

Recommendation: Add a `protocolFee` variable, increment it every time that there is a fee that the protocol should claim and add a separate function that allows the protocol owner to withdraw fee whenever they want.

t3rn: Fixed in commit 590e0e8e.

Cantina Managed: Fix verified.

3.3.3 `checkIsRefundable` returns `true` when protocol is halted

Severity: Low Risk

Context: remoteOrder.sol#L406

Description: `checkIsRefundable` is a public function and it is allowed to be called by any external system that wants to integrate with t3rns infrastructure. The `checkIsRefundable` function checkes whether an order is refundable for certain user:

```
function checkIsRefundable(
    bytes32 orderId,
    uint256 orderTimestamp,
    address rewardAsset,
    uint256 maxReward
) public view returns (bool) {
```

Though it fails to check whether the contract is on a halted state, were refunds are effectively paused:

```
function claimRefundV2(
    uint32 orderNonce,
    address rewardAsset,
    uint256 maxReward,
    uint256 orderTimestamp,
    bytes32 _batchPayloadHash,
    bytes memory _batchPayload
) public payable isOn {
```

Therefore, it will return the wrong boolean as in fact the order can't be refunded while `isOn` is true.

Recommendation: Add a check so that if `isOn` is set to true, then `checkIsRefundable` will return false.

t3rn: Fixed in commits f53a453 and c006309.

Cantina Managed: Fix verified.

3.3.4 Forcefully bypassed event emission on empty payloads

Severity: Low Risk

Context: escrowGMP.sol#L173

Description: Inside the `commitEscrowBeneficiaryPayload` function, the result of `escrowOrdersPayloadHash[sfxId]` is checked to not be 0 twice, the first time is just returns false and the second one (which won't be triggered), emits the correct event and returns false:

```
function commitEscrowBeneficiaryPayload(bytes32 sfxId, address beneficiary) external onlyAttesters returns
↪  (bool) {
    // Check if the payload exists and return false if it doesn't
    if (escrowOrdersPayloadHash[sfxId] == 0) {
        return (false);
    }

    // Update the payment payload (hash of the payload)
    bytes32 currentHash = escrowOrdersPayloadHash[sfxId];
    if (currentHash == bytes32(0)) {
        emit GMPExpectedPayloadNotMatched(sfxId, address(0), currentHash);
        return (false);
    }
```

Recommendation: Remove the first check so that the event can be emitted for empty hashes:

```diff
  function commitEscrowBeneficiaryPayload(bytes32 sfxId, address beneficiary) external onlyAttesters returns
  ↪  (bool) {
-     // Check if the payload exists and return false if it doesn't
-     if (escrowOrdersPayloadHash[sfxId] == 0) {
-         return (false);
-     }

      // Update the payment payload (hash of the payload)
      bytes32 currentHash = escrowOrdersPayloadHash[sfxId];
      if (currentHash == bytes32(0)) {
          emit GMPExpectedPayloadNotMatched(sfxId, address(0), currentHash);
          return (false);
      }
```

t3rn: Fixed in commit dc7bf8f.

Cantina Managed: Fix verified.

3.3.5 `msg.value` can be forwarded when the reward token is not ETH

Severity: Low Risk

Context: remoteOrder.sol#L214

Description: When calling `orderMemoryData` it checks for `msg.value` or it transfers the erc20 tokens from the sender. Both cases are handled, but there is a case missing to check for and it is when the rewardAsset is not ETH, but `msg.value` is being sent:

```
if (rewardAsset == address(0)) {
    require(msg.value == maxReward, "RO#7");
    require(amount < maxReward, "RO#7");
} else {
    IERC20(rewardAsset).safeTransferFrom(msg.sender, address(this), maxReward);
}
```

Recommendation: Add a check when `rewardAsset` is not ETH for `msg.value` to be 0.

t3rn: Fixed in commit 85b3fd5.

Cantina Managed: Fix verified.

3.3.6 Incorrect Rounding Direction in Fee Calculation

Severity: Low Risk

Context: remoteOrder.sol#L229.

Description: The protocol fee calculation uses incorrect rounding, causing the Protocol Fee Amount to be rounded down. As a general rule, protocol fee calculations are usually rounded up on DeFi protocols.

Recommendation: Adjust the rounding logic to ensure protocol fees are rounded up, aligning with industry best practices:

```
function calcProtocolFee(uint256 amount) public view returns (uint256) {
    return (amount * currentProtocolFee + 1e6 - 1) / 1e6;
}
```

t3rn: Fixed in commit 733f1f74.

Cantina Managed: Fix verified.

3.3.7 Consider moving `_disableInitializers` to Constructor for Consistency

Severity: Low Risk

Context: claimerGMPV2.sol#L49

Description: The `_disableInitializers` function, inherited from `Initializable`, is currently callable via `disableInitializers`. To streamline initialization security, consider moving this call directly to the constructor, eliminating the need for an external function. This approach aligns with the pattern used in other contracts in the codebase, such as `remoteOrder`.

Recommendation: Invoke `_disableInitializers` within the constructor instead of requiring a separate function call, ensuring consistency and preventing unnecessary external interactions.

t3rn: Fixed in commit 5e81dcc.

Cantina Managed: Fix verified.

3.3.8 `claimerGMPV2` `generateId` Function Does Not Include `sourceId`

Severity: Low Risk

Context: claimerGMPV2.sol#L62-L64

Description: The `generateId` function currently hashes only the `requester` and `nonce`, but it does not incorporate `sourceId`. This results in incorrect getter outputs.

Recommendation: Modify the function to include `sourceId` in the hash computation.

t3rn: Fixed in commit b2c37089.

Cantina Managed: Fix verified.

3.3.9 Improve Pause Mechanism for More Granular Control

Severity: Low Risk

Context: remoteOrder.sol#L148-L154

Description: The current pause mechanism on `remoteOrder` is binary, meaning the entire contract is either fully operational or completely halted. There is no selective pausing for specific functions (e.g., allowing claims while preventing new orders). This lack of flexibility may limit the contract's ability to respond to different situations effectively.

Recommendation: Consider implementing the pause mechanism with a more granular approach.

t3rn: Fixed in commit c006309.

Cantina Managed: Fix verified.

3.3.10 Ensure `executionCutOff` is always less than `orderTimeout` across chains

Severity: Low Risk

Context: remoteOrder.sol#L287

Description: The `setExecutionCutOff` function allows the owner to set the executionCutOff value. However, since `executionCutOff` and `orderTimeout` exist on different chains, there is a potential situation where `executionCutOff` might be set higher than `orderTimeout` on the source chain, leading to unintended executions or inconsistencies across chains.

Recommendation: Ensure that, during every deployment or configuration update, `executionCutOff` is always set to a value smaller than `orderTimeout`. This can be enforced through:

- Off-chain validation and deployment scripts.
- Explicit checks in setter functions (`setExecutionCutOff`, `setOrderTimeout`) by enforcing min/max value constraints at the contract level.

t3rn: Fixed in commit 2d123168.

Cantina Managed: Fix verified.

3.3.11 Missing cap in fees

Severity: Low Risk

Context: remoteOrder.sol#L157

Description: The function `setCurrentProtocolFee` is missing a cap so that it can not be set to over a certain amount. Usually, as a placeholder, the max amount can be set to 100%:

```
function setCurrentProtocolFee(uint256 _protocolFee) external onlyOwner {
    currentProtocolFee = _protocolFee;
}
```

Recommendation: Add a maximum protocol fee.

t3rn: Fixed in commit cf06acb3.

Cantina Managed: Fix verified.

3.4 Informational

3.4.1 Minor improvements to code and comments

Severity: Informational

Context: *(See each case below)*

Description/Recommendation:

1. escrowGMP.sol#L4-L7 - The imported OpenZeppelin contracts (ReentrancyGuard, OwnableUpgradeable, and Initializable) are not used in the code.
2. escrowGMP.sol#L15-L16 - Deprecated mappings (`escrowCallsPayments` and `escrowCallExists`) are still present in the codebase. Consider removing them if they are no longer needed.
3. escrowGMP.sol#L100-L104 - The `onlyOwner` modifier should be applied directly to the external functions rather than being enforced within an internal function. This would improve readability and make access control clearer at the function level.
4. escrowGMP.sol#L255 - Consider defining constants for `bytes32(0)`, `bytes32(1)`, and `bytes32(2)` to improve code maintainability, readability, and reduce redundancy.
5. avpBatchSubmitter.sol#L4-L8 - The imported OpenZeppelin contracts (MerkleProof, OwnableUpgradeable, and Initializable) are not used in the code.
6. avpBatchSubmitter.sol#L40 - The `initialize` function in `avpBatchSubmitter` lacks address validation, whereas remoteOrder includes it. Consider adding validation to initialize for consistency and security.
7. avpBatchSubmitter.sol#L57 - Consider adding events and address validation to all setter functions to enhance transparency, traceability, and input safety.
8. avpBatchSubmitter.sol#L126 - For improved readability, consider moving the BatchData struct to the beginning of the contract.
9. avpBatchSubmitter.sol#L396, remoteOrder.sol#L251, remoteOrder.sol#L497-L526 - Remove comment.

10. remoteOrder.sol#L4-L10 - The imported OpenZeppelin contracts (Context, OwnableUpgradeable, and Initializable) are not used in the code.
11. remoteOrder.sol#L65-L68 - Multiple fields, such as `nonce`, that use uint types may not need to be indexed. Consider removing the indexed keyword for these fields to optimize gas usage.
12. remoteOrder.sol#L120 - Consider adding events and address validation to all setter functions to enhance transparency, traceability, and input safety. Additionally, no event is emitted when isHalted changes via turnOnHalt/turnOffHalt.
13. remoteOrder.sol#L195 - The `isOn` modifier is currently duplicated when called via the `order` function. Consider either removing it from order or removing it from `orderMemoryData` and making the function internal to reduce redundancy and improve code quality.
14. remoteOrder.sol#L223 - Emits `RemoteOrderCreated` with limited data, while the wrapper order function emits `OrderCreated` with full details.
15. remoteOrder.sol#L229 - Consider replacing magic numbers with constants throughout the codebase. This will enhance readability, maintainability, and reduce the risk of errors.
16. remoteOrder.sol#L402 - The `isOn` modifier is currently duplicated when called via the `claimRefund` function. Consider either removing it from `claimRefund`.
17. remoteOrder.sol#L342-L359 - The check if (address(claimerGMP) == address(0)) return false; is missing.

t3rn: Fixed in commit c2e770c7.

Cantina Managed: Fix verified.