

SECURITY ANALYSIS

by Pessimistic

Abstract	2
Disclaimer	2
Summary	2
General recommendations	2
Project overview	3
Project description	3
Codebase update	3
Audit process	4
Manual analysis	5
Critical issues	5
Medium severity issues	6
M01. Unaccounted L2 fee components (fixed)	6
M02. Front-running transactions (fixed)	6
M03. Griefing of relayer (addressed)	6
Low severity issues	7
L01. Check of balance before execution (fixed)	7
L02. Users with attached contract cannot receive NFTs (fixed)	7
L03. Unaccounted gas for signature hashing (fixed)	7
Notes	8
N01. Verified executor address must be enforced (commented)	8
N02. The users can reattach different contracts to their addresses (commented)	8

ABSTRACT

This report evaluates the security of the smart contracts in the [Tangem Gassless Transactions](#) project. The objective was to identify and document security vulnerabilities within the platform's smart contracts.

DISCLAIMER

This audit makes no warranties or guarantees as to the security of the code. A single audit is insufficient to ensure complete security. We strongly recommend conducting multiple independent audits and implementing a public bug bounty program to enhance smart contract security. Additionally, this security audit should not be construed as investment advice.

SUMMARY

This report assesses the security of the [Tangem Gasless Transactions](#) smart contracts. The [audit process](#) is detailed in the following section.

The audit identified several medium severity issues: [Unaccounted L2 fee components](#), [Front-running transactions](#), [Griefing of relayer](#). Also, two low severity issues were found.

The overall code quality is good.

After the initial audit, the codebase was [updated](#). [M03](#) issue was addressed and commented on. All other issues were fixed.

GENERAL RECOMMENDATIONS

We do not have any further recommendations.

PROJECT OVERVIEW

Project description

For this audit, we reviewed the [Tangem Gasless Transactions](#) project from a public GitHub repository, commit [e6e1b0c84f38785a7d75f51da16c919a8f59e2fa](#).

The scope of the audit included everything on that commit.

The provided documentation included:

- Gasless_contracts_requirements.pdf (sha1sum [b7b8f54cf6f06069392ac24b1324332e65db9d45](#)),
- Gasless contracts description.pdf (sha1sum [97cade2b284250fa03f3374b82479f7a3ef77ad7](#)),
- Gasless_transaction_flow.pdf (sha1sum [28bfa3601bc5e7158a10a528b4bb05eae5fa0956](#)).

The contracts will be deployed to:

- [Ethereum](#);
- [Arbitrum](#);
- [Base](#);
- [BSC](#);
- [Polygon POS](#).

All 16 tests pass successfully. The code coverage is 100%.

The total lines of code (LOC) of audited sources is 241.

Codebase update

For the recheck, the developers provided us with a new commit [b5ca9fcd4b1d7e2a95cd229efa5aec504b1dd446](#). One medium-severity issue was addressed, and all other issues were fixed.

All the 16 tests passed. The code coverage is 95.21%.

AUDIT PROCESS

The audit was conducted from January 19, 2026, to January 21, 2026.

We began by reviewing the provided materials, then met with the development team for a project overview.

Throughout the audit, we maintained close communication with the developers to clarify ambiguous or concerning aspects of the code.

We performed a comprehensive manual review of all contracts within the audit scope, verifying the logic and security properties. Among others, we specifically verified the following areas and did not identify issues related to them:

- User funds are protected from project owner and relayers;
- The contract is protected against replayed transactions (including cross-chain replay);
- Transaction fees are calculated correctly.

We analyzed the project with the following automated tools:

- AI scanner [AuditAgent](#);
- AI scanner [Savant Chat](#);
- [Semgrep](#) rules for smart contracts.

We executed the test suite and calculated code coverage.

All identified issues — whether detected manually or through automated tools — were compiled into the private report.

Following the initial audit, the developers provided us with the updated [codebase](#), which contained fixes. We performed the recheck on January 23-26, 2026. One issue of medium severity was commented on and marked as addressed. All other issues were fixed.

MANUAL ANALYSIS

All contracts were manually inspected to validate their logic and security. Additionally, automated analysis results were manually verified. Confirmed issues are documented below.

Critical issues

Critical issues pose severe risks to project security, potentially leading to fund loss or other catastrophic failures. Contracts should not be deployed until such issues are resolved.

No critical issues were identified during the audit.

Medium severity issues

Medium severity issues may impact the project's current functionality. This category includes bugs, potential revenue loss, operational inefficiencies, and risks related to improper system management. We strongly recommend addressing these issues.

M01. Unaccounted L2 fee components (fixed)

The fee calculation accounts for L2 execution gas only, but Arbitrum and Base transactions also include an L1-related fee component (Arbitrum parent-chain calldata [poster fee](#), and Base L1 [security fee](#)). This can underestimate the true transaction cost, leading to undercharging, relayer losses, or failed submissions due to insufficient fees.

| The issue has been fixed and is not present in the latest version of the code.

M02. Front-running transactions (fixed)

`feeReceiver` is provided as calldata to `executeTransaction` but is not part of the signed [EIP-712](#) payload. Since `Tangem7702GaslessEntryPoint.executeTransaction` and `Tangem7702GaslessExecutor.executeTransaction` are publicly callable, anyone can copy a relayer's pending transaction and front-run it with the same `gaslessTx` and signature but a different `feeReceiver`, redirecting the fee. The relayer's transaction then reverts due to nonce usage, so the relayer pays gas for a reverted transaction.

| The issue has been fixed and is not present in the latest version of the code.

M03. Griefing of relayer (addressed)

An attacker can craft transactions that pass the relayer's off-chain simulation but revert when submitted on-chain, causing the relayer to pay gas with no compensation. The attacker can increase the relayer's loss by consuming large gas before reverting (for example, long loops or large revert data), and can repeat this to drain relayer funds and reduce service availability.

Comment from the developers:

The service is initially available only to the users of our app. Users won't get any profit for doing this attack. We are going to monitor usage for illicit activity and deny service to malicious users. We're also setting a max gas limit to limit the damage.

Low severity issues

Low severity issues do not immediately affect project operations but may introduce risks in future iterations. We recommend resolving these issues or providing justification for the chosen implementation.

L01. Check of balance before execution (fixed)

`Tangem7702GaslessExecutor.executeTransaction` reverts if the current `feeToken` balance is below `maxTokenFee` before running the user call. This can block valid cases where the `executor` receives `feeToken` during the call (for example, swap/transfer-in).

| The issue has been fixed and is not present in the latest version of the code.

L02. Users with attached contract cannot receive NFTs (fixed)

`Tangem7702GaslessExecutor` does not implement `onERC721Received` and `onERC1155Received`. As a result, `safeTransferFrom` transfers revert, so the user cannot receive NFTs.

| The issue has been fixed and is not present in the latest version of the code.

L03. Unaccounted gas for signature hashing (fixed)

At lines 60-62 of the `Tangem7702GaslessExecutor` contract, gas accounting starts only after `_verifyGaslessTransaction`, which consumes gas to hash the metatransaction data. The gas usage can be large depending on user-provided data.

| The issue has been fixed and is not present in the latest version of the code.

Notes

N01. Verified executor address must be enforced (commented)

It is important that the **Tangem7702GaslessExecutor** implementation attached to the user account is the verified and audited deployment. The system relies on a specific executor address (expected to be hardcoded in the wallet application), so an incorrect or unverified address can route executions through unintended code. This can undermine the security assumptions of the audit and expose users to malicious or unsafe executor logic.

Comment from the developers:

The address will be hardcoded into the app, ensuring the best security in this regard.

N02. The users can reattach different contracts to their addresses (commented)

A user can reattach (set code) different contract implementation to his address and thereby change or reset the nonce used by **Tangem7702GaslessExecutor**. This can invalidate previously signed transactions and can cause self-inflicted denial of service by desynchronizing the expected **nonce** or replacing the executor logic.

Comment from the developers:

- We're not relying on nonce anywhere outside of the gasless transaction verification itself, and not storing transactions for the future;
- We rely on Tangem7702GaslessEntryPoint contract to be sure what contract is attached to the user account;
- This self-inflicted denial of service can only intentional, accidental is not possible due to usage Custom Storage Layout.

This analysis was performed by **Pessimistic**:

Yhtyyar Sahatov, Security Engineer

Daria Korepanova, Senior Security Engineer

Evgeny Marchenko, Senior Security Engineer

Irina Vikhareva, Project Manager

Alexander Seleznev, CEO

January 26, 2026