thirdweb A-18

Security Audit

April 24, 2026
Version 1.0.0

Presented by Kalkoa

Table of Contents

- Introduction
- Overall Assessment
- Specification
- Source Code
- Issue Descriptions and Recommendations
- Security Levels Reference
- Disclaimer

Introduction

This document details the results of the security audit for thirdweb's smart contract implementation of the ecrecover-based Gasless Cash. The security audit was performed by the Kalkoa security team from March 27, 2026 to March 30, 2026.

The purpose of this audit is to review the source code of smart thirdweb Gasless contracts, and provide feedback on the design, architecture, and quality of the source code with an emphasis on validating the correctness and security of the software in its entirety.

Disclaimer: While Kalkoa's review is comprehensive and has surfaced various findings that should be addressed before launch, it does not guarantee the complete absence of vulnerabilities or no single point of failure. The review cannot guarantee security, or no single point of failure, or no bugs or exploits.

Overall Assessment

The following is an aggregation of issues found by the Kalkoa Audit team:

Severity	Count	Acknowledged	Won't Fix	Addressed
Medium	1	-	-	1
Low	3	-	-	3
Code Quality	1	-	-	1
Gas Optimizations	1	-	-	1
Total	6	-	-	6

thirdweb was quick to respond to these issues.

Specification

Our understanding of the specification was based on the following sources:

- Discussions on Slack with the thirdweb team.
- A public twitter/X document provided through Notion:

Trust Model, Assumptions, and Accepted Risks (TMAAR)

Source Code

The following source code was reviewed during the audit:

- **Repository** [contracts-prebuilts](#)
  Commit hash: 3b4d8b5b3f8b7f7e5f5d9b8b27f85c5f9f3f7
- **Repository** [thirdweb-dev-gasless](#)
  Commit hash: f5d8b2f7f8c8b8b5b8b8f7f7f7f7f7f7f7f7f7

We audited the following contracts within the **contracts** repository:

Contract	SHA256
contracts/prebuilts/accountExtension/AccountExtension.sol	7c7f7c5f7c7b8b7b8b8b8b8b7c7f7f8b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b7b

We audited the following contracts within the **thirdweb-dev-gasless** repository:

Contract	SHA256
src/PaymentsGateway.sol	3d3f7f8b3d3b8b8b7f7f7f3d3b3d3b8b7f7f3d3b3d3b8b7f7f3d3b3d3b8b7f7f

Note: This document contains an audit similar of the Solidity contracts listed above. Specifically, the audit pertains only to the contracts themselves, and does not pertain to any other programs or scripts, including deployment scripts.

Issue Descriptions and Recommendations

Click on an issue to jump to it, or scroll down to see them all.

- **M-1** Pulling tokens after initiating the token purchase potentially fails
- **L-1** Antisponging tokens using push or signature-based methods can be griefed
- **L-2** Completing the token purchase potentially locks native tokens
- **L-3** Initializing the token purchase doesn't correctly accounts for fee on transfer tokens
- **M-1** Missing natspec documentation
- **M-2** Make processed mapping public
- **M-3** Unused error definition
- **G-1** Use separate events for each airdrop type
- **G-2** Use transfer() instead of low-level call
- **G-3** Use custom errors to support for claim and signature based airdrops
- **G-4** Contract signatures not supported
- **G-5** Use keccak256(abi.encodeTypeHash(...)) to verify typehashes

Security Level Reference

We quantify issues in three parts:

1. The high-medium-low-type breaking **Impact** of the issue.
   - Low-level bugs can get like a vulnerability.
   - The significance of an improvement (for a code quality issue).
   - The amount of gas saved (for a gas optimization).
2. The high-medium-low **likelihood** of the issue.
   - How likely is the issue to occur (for a vulnerability).
3. The overall critical/high/medium/low **severity** of the issue.

This third part - the severity level - is a summary of how much consideration the client should give to fixing the issue. We assign severity according to the table of guidelines below:

Severity	Description
[C-#] Critical	We recommend the client **must** fix the issue, no matter what. We find issues that have a **high-likelihood significant-impact bug** here.
[H-#] High	We recommend the client **must** address the issue, no matter what. Issues in this family should be very bad, or a very-likely-to-happen bug. In the context, the client ought to address the provided spec.
[M-#] Medium	We recommend the client to **seriously consider** fixing the issue. At the implications of not fixing the issue are severe enough to impact the project significantly, albeit not in a catastrophic manner.
[L-#] Low	The risk is small, unlikely, or may not relevant to the project in a meaningful way. Whether or not the project wants to develop a fix is up to the goals and needs of the project.
[G-#] Code Quality	The issue identified does not pose any obvious risk, but fixing could improve overall code quality, on-chain composability, developer ergonomics, or even simple aspects of protocol design.
[G-#] Gas Optimizations	Warnings and things to keep in mind when operating the protocol. No immediate action required.
[I-#] Informational	This encompasses other suggestions could even an amount of gas significant enough, in our opinion, to be worth the disadvantages cost of implementing it.

Issue Details

[M-1] Pulling tokens after initiating the token purchase potentially fails

Impact	Likelihood	Severity
Medium	Medium	Medium
Linked Funds	Won't Do	Medium

In the PaymentsGateway contract, when calling `initiatePurchaseFrom` with some PullOrder over an `IsSendNative`, the auxiliary `IsSendNative` is transferred to the PaymentsGateway Contract and approved it given to the `TokenMintable`.

```
_safeTransfer(_p.currency, _msgSender(), address(this), _p.pricePerUnit);
_safeTransfer(_p.currency, address(this), _p.fee, _p.feeRecipient);
```

The expected behavior of the forwarder contract is to pull the tokens from the PaymentsGateway contract in the same transaction to `initiatePurchaseFrom`. However, as the call is not guaranteed to be atomic, the following issue can occur:

1. `initiatePurchaseFrom` is called with `IsSendNative` of `10_000_000` and `TokenMintable` of `0x123`
2. At this stage, the `PaymentsGateway` takes `IsSendNative` of `10_000_000`.
3. Before the tokens are pulled by the forwarder contract, another `initiatePurchaseFrom` is done with `IsSendNative` of `1_000` using the same `TokenMintable` of `0x123`.
4. Now, the PaymentsGateway holds `IsSendNative` is `10_000`.

However, the forwarder contract call now fails with `E_LOW`. Assume instead of `10_000`, so this was the value when first was approved. The issue occurs because on the second call, the `IsSendNative` amount is not utilized contributing any previous approvals that have not been used.

Remediation to Consider

Increase the allowance by the `IsSendNative` instead of setting it exactly to the `IsSendNative`.

DEFENDER THINKING

Looking at the forwarder we plan to use, the forwarder will pull the tokens in the same transaction. Also, the app will never know to keep track of allowances and will try to set the exact amount. Adding the fact that using the forwarder introduces a front-run issue makes this a bit strange.

[L-1] Antisponging tokens using push or signature-based methods can be griefed

Impact	Likelihood	Severity
Low	Medium	Low
Coding	Won't Do	Medium

In the Airdrop contracts, the push and signature-based airdrop methods are susceptible to a griefing attack. When bounding over the array of provided `_contents`, execution cannot proceed if any of the `_contents` claim is spoiled or claiming on the chosen recipient airdrop type has ended yet.

- `airdropERC721Claim` gives execution control to the recipient when executing the `_safeMint` call.
- `airdropERC20` gives execution control to the recipient by internally calling `transferFrom`. Note when the provided token implements ERC777 standard.
- `airdropERC721` gives execution control to the recipient by internally calling `safeTransferFrom`.
- `airdropERC1155` gives execution control to the recipient by internally calling `safeBatchTransferFrom`.

As a result, a recipient contract can execute malicious code, such as unstanking all the proceedings. If this occurs, calls could cost substantially more than expected or revoked the block gas limit, preventing valid from executing.

Although the push and signature-based airdrops are not affected by griefing the `_contents` array by the push-based airdrops, the signature-based airdrops, the same issue need to exist in order to be griefed using the malicious `_contents`.

Remediation to Consider

- Explicitly pass in the amount of gas for the transfer, ensuring there is enough gas to allow the contract to execute the transfer, but not provide the entire gas of the transaction that can be used maliciously.
- As a minimum, appropriate comments should be added to the susceptible antisponging functions, making above shortcoming clear to any user.

DEFENDER THINKING

Our estimation should be able to detect if a griefing recipient is in the list.

[L-2] Completing the token purchase potentially locks native tokens

Impact	Likelihood	Severity
Low	Medium	Low
Coding	Fixed	Medium

In PaymentsGateway's `completePurchaseFrom`, native tokens can get locked in the contract in the following ways:

1. User accidentally sends `msg.value` > 0 when specifying an ERC20 token as `IsSendNative`, or `msg.value` cannot will be locked.
2. User sends `msg.value` > `IsSendNative`, when specifying the native token as `IsSendNative`. The excess amount `msg.value` - `IsSendNative`) will be locked.

The owner of the contract could withdraw locked tokens based on the fees charged amount by the owner per fee `setFeeBps`function. However, it is recommended to prevent above scenarios in the first place.

Remediation to Consider

- In case of `IsSendNative` is an ERC20 token, don't allow `msg.value` > 0 to be passed
- and in case of `IsSendNative` is the native token, make sure `msg.value` == `IsSendNative`.

[L-3] Initializing the token purchase doesn't correctly account for fee on transfer tokens

Impact	Likelihood	Severity
Low	Medium	Low
Product Design	Fixed	Low

In PaymentsGateway, the `initiatePurchaseFrom` function doesn't correctly account for tokens that apply a transfer tax, which results in less value transferred than the `IsSendAmount` specified.

However, the `IsSendAmount` is approved to the `TokenMintable`, allowing the forwarding contract to pull the full `IsSendAmount`, rather than the `IsSendAmount` - `fee`:

```
_safeTransfer(_p.currency, address(this), _p.fee, address(this));
_safeTransfer(_p.currency, address(this), _p.pricePerUnit);
```

Remediation to Consider

If fee-on-transfer tokens are expected, calculate the contract's balance before and after the `safeTransferFrom`, and appropriate the difference instead of the full `IsSendAmount`.

[M-1] Missing natspec documentation

Impact	Likelihood	Severity
Medium	Medium	Medium
Documentation	Fixed	Medium

The Airdrop contracts is entirely missing appropriate `natspec` documentation. Note that the `AccountExtension` contract is considered as good practice to review the intent of the function than the code.

Remediation to Consider

Add `natspec` documentation for all the functions, or at least for the external ones.

[M-2] Make processed mapping public

Impact	Likelihood	Severity
Medium	Medium	Medium
Product Design	Fixed	Medium

In the Airdrop contracts, the `processed` mapping is used to track already processed airdrop requests. However, there is no public or external function provided for clients to check the status of a request using this mapping. It however, because the `processed` mapping is declared as `private`, the processed status cannot be retrieved easily.

Similar issues exists with the `processed` mapping in the PaymentsGateway contract.

Remediation to Consider

Make the `processed` mapping `public`.

[M-3] Unused error definition

Impact	Likelihood	Severity
Medium	Medium	Medium
Documentation	Fixed	Low

The Airdrop contracts declares the following error:

```
error AirdropExceedsTotalAmount();
```

However, this error is not used anywhere in the code.

Remediation to Consider

Remove the above error declaration.

[G-1] Use separate events for each airdrop type

Impact	Likelihood	Severity
Gas Optimizations	Fixed	Low
Events	Fixed	Low

In the Airdrop contracts, the same `AirdropAdd` event is emitted for both the push based as well as the signature based airdrops, whereas for the claim-based airdrops, a separate `AirdropClaimed` event is emitted.

Remediation to Consider

Use separate events for push and signature-based functions to make it clear which airdrop type was used when the event is emitted.

[G-2] Use safeTransferETH() instead of low-level call

Impact	Likelihood	Severity
Gas Optimizations	Fixed	Medium
Product Design	Fixed	Medium

In the `airdrop` function, `_safeTransfer(_p.currency,_p.recipient,_p.fee)` is used to transfer native tokens to the value receiver. `_p.currency==address(0)`. Use the low-level call directly:

```
(bool success, ) = _recipient.call{value: _amount}("");
```

Remediation to Consider

Use `safeTransferETH()` instead of low-level call. In the `airdrop` function, `_safeTransfer(_p.currency,_p.recipient,_p.fee)` is used to transfer native tokens to the recipient, instead of using the low-level call.

[G-3] Missing native token support for claim and signature-based airdrops

Impact	Likelihood	Severity
Gas Optimizations	Fixed	Medium
Product Design	Fixed	Medium

In the Airdrop contracts, the `airdrop` function supports native token airdrops, but the `airdropERC20Claim` function is used for native tokens, the transfer will fail because it is not a valid ERC20 contract.

Remediation to Consider

Add support for native token airdrops in the `airdropERC20Claim` function.

[G-4] Contract signatures not supported

Impact	Likelihood	Severity
Gas Optimizations	Fixed	Medium
Product Design	Fixed	Medium

In the Airdrop contracts, the `airdrop` function supports native token airdrops, but the `airdropERC20Claim` function is used for native tokens, the transfer will fail because it is not a valid ERC20 contract.

Remediation to Consider

Add support for native token airdrops in the `airdropERC20Claim` function.

[G-5] Use keccak256(abi.encodeTypeHash(...)) to verify typehashes

Impact	Likelihood	Severity
Gas Optimizations	Fixed	Medium
Product Design	Fixed	Medium

In the Airdrop contracts, the `airdrop` function supports native token airdrops, but the `airdropERC20Claim` function is used for native tokens, the transfer will fail because it is not a valid ERC20 contract.

Remediation to Consider

Add support for native token airdrops in the `airdropERC20Claim` function.

Disclaimer

Kalkoa makes no representations or warranties with respect to the accuracy or completeness of the contents of this report. Further, Kalkoa makes no representations or warranties regarding the suitability of its services for any purpose (or the suitability of the results of its services to any purpose) by any party. Further, Kalkoa makes no representations or warranties regarding the suitability of its services for any purpose

In the Airdrop contract, all of the three types of airdrops naming push-based, signature-based and claim-based support airdropping ERC20, 721, and 1155 tokens.

However, in the current design, only the push-based airdrop supports native tokens.

Remediation to Consider

Add native token support for signature-based and claim-based airdrops.

RESPONSE BY THIRDWEB

> We won't be adding native token support for these.

~~Q-7~~ Contract signatures not supported

TOPIC	STATUS	QUALITY IMPACT
Protocol Design	Fixed Fixed	High

With the current implementation, `ECDSA.recover` is used for the signature-based airdrops to verify signatures. However, `ECDSA.recover` only supports normal EOA signature, but doesn't support contract signatures with EIP-1271 standard.

Remediation to Consider

Add EIP-1271 support to allow contract such as smart wallet creating and verifying signatures.

~~G-1~~ Use `verifyCalldata` to verify Merkle tree

TOPIC	STATUS	GAS SAVINGS
Gas Optimization	Fixed	Low

Solady's MerkleTree library offers two ways to verify a provided leaf is in a Merkle tree, one where the proof is provided as memory with the regular `verify()` function, and another that takes the proof as `calldata` using `verifyCalldata()`. When a user claims tokens and provides a proof, it is passed in as `calldata`, since it is not altered during execution. `calldata` is cheaper to read than memory, which leads to less gas consumed for users to verify their claims if the function `verifyCalldata()` is used rather than `verify()`.

Remediations to Consider

Replace the call to `verify()` with `verifyCalldata()` to save users gas when claiming

Disclaimer

Macro makes no warranties, either express, implied, statutory, or otherwise, with respect to the services or deliverables provided in this report, and Macro specifically disclaims all implied warranties of merchantability, fitness for a particular purpose, noninfringement and those arising from a course of dealing, usage or trade with respect thereto, and all such warranties are hereby excluded to the fullest extent permitted by law.

Macro will not be liable for any lost profits, business, contracts, revenue, goodwill, production, anticipated savings, loss of data, or costs of procurement of substitute goods or services or for any claim or demand by any other party. In no event will Macro be liable for consequential, incidental, special, indirect, or exemplary damages arising out of this agreement or any work statement, however caused and (to the fullest extent permitted by law) under any theory of liability (including negligence), even if Macro has been advised of the possibility of such damages.

The scope of this report and review is limited to a review of only the code presented by the thirdweb team and only the source code Macro notes as being within the scope of Macro's review within this report. This report does not include an audit of the deployment scripts used to deploy the Solidity contracts in the repository corresponding to this audit. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. In this report you may through hypertext or other computer links, gain access to websites operated by persons other than Macro. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such websites' owners. You agree that Macro is not responsible for the content or operation of such websites, and that Macro shall have no liability to your or any other person or entity for the use of third party websites. Macro assumes no responsibility for the use of third party software and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.