Safe Edges

AUDIT REPORT

2025

SAFEEDGES.IN

Clique Audit Report

Reviewed by: SafeEdges
Prepared for: Clique
Review Dates: `3/06/2025 - 05/07/2025`

Protocol Summary

Clique Audit

In-Scope:

`Reviewed contracts hash:`

`SigDispersal.sol: 3caeb0e4aeb0afcf085af1cf713ed5f1f13b539b7689db9ea10a8b73619325c6`

No.	Finding Name	Severity	Status
H-01	Missing `batchEnabled` check prior to claiming in `claim` function	High	Resolved
L-01	Mistakenly sent ETH to `claim` function will remain stuck forever	Low	Resolved
I-01	Implement checks prior to any call to save gas	Informational	Resolved
I-02	Lack of deadline or time constraint on airdrop duration	Informational	Resolved
I-03	Missing zero address check	Informational	Resolved

[H-01] Missing `batchEnabled` Check Prior to Claiming in `claim` Function

Description: The `claim` function is designed to allow users to claim their airdropped tokens. However, it does **not** include a check for the `batchEnabled` mapping, which is intended to verify whether a particular airdrop batch is currently active or not. This omission may allow users to successfully claim tokens from a disabled batch, provided they possess a valid signature from the authorized `signer`. As a result, the `batchEnabled` mapping becomes redundant (dead code) and offers no protection.

We recommend using batchId as the key in the batchEnabled mapping instead of root. This will make the batch enable/disable functionality meaningful and ensure proper validation before claims.

Impact:

High – Even if the `PROJECT_ADMIN` disables a batch, users can still claim tokens using valid signatures, rendering batch control ineffective.

Mitigation: Add a check to ensure the batch is enabled before proceeding with the claim. Suggested code change:

```
function claim(bytes calldata _signature, bytes32 _batch, uint256 _amount, address
_onBehalfOf)
    external
-    payable
    whenActive
{
+    if (!batchEnabled[_batch]) {
+        revert NotActive();
+    }

    address _signer = signer;

    _signatureCheck(_amount, _onBehalfOf, _signature, _signer, _batch);

    uint256 _claimHeight = claimed[_batch][_onBehalfOf];
    if (_claimHeight > 0) {
        revert AlreadyClaimed();
    }

    claimed[_batch][_onBehalfOf] = block.number;

    IERC20(token).safeTransferFrom(vault, _onBehalfOf, _amount);

    emit AirdropClaimed(_batch, _onBehalfOf, _amount);
}
```

RESOLVED

[L-01] Mistakenly Sent ETH to `claim` Function Will Be Stuck Forever

Description: The `claim` function is marked `payable`, but the function does not use or handle any native ETH. Moreover, the contract lacks a `receive()` or `fallback()` function. As a result, if users accidentally send ETH with their claim, the funds will become permanently stuck in the contract.

Impact: Low – Affects users only if they mistakenly send ETH during a claim call.

Mitigation: Remove the `payable` modifier from the `claim` function, since it serves no purpose.

```
function claim(bytes calldata _signature, bytes32 _batch, uint256 _amount, address
_onBehalfOf)
    external
-    payable
    whenActive
{ ... }
```

RESOLVED

[I-01] Reorder Checks to Save Gas

Description: In the `claim` function, `_signatureCheck` is called before verifying whether the user has already claimed the airdrop. This means unnecessary gas is consumed on signature verification even when the user is ineligible due to an existing claim.

Impact: Informational, but relevant for gas optimization.

Mitigation: Reorder the checks to first validate the claim eligibility, then verify the signature:

```
function claim(bytes calldata _signature, bytes32 _batch, uint256 _amount, address
_onBehalfOf)
    external
    whenActive
{
+    uint256 _claimHeight = claimed[_batch][_onBehalfOf];
+    if (_claimHeight > 0) {
+        revert AlreadyClaimed();
+    }

    address _signer = signer;

    _signatureCheck(_amount, _onBehalfOf, _signature, _signer, _batch);

-    uint256 _claimHeight = claimed[_batch][_onBehalfOf];
-    if (_claimHeight > 0) {
-        revert AlreadyClaimed();
-    }

    claimed[_batch][_onBehalfOf] = block.number;

    IERC20(token).safeTransferFrom(vault, _onBehalfOf, _amount);

    emit AirdropClaimed(_batch, _onBehalfOf, _amount);
}
```

RESOLVED

[I-02] Lack of Deadline or Expiry Mechanism for Airdrop Batches

Description: Currently, once a batch is enabled, it remains active indefinitely unless explicitly disabled by the `PROJECT_ADMIN`. However, the disabling process does **not** impact the claim function unless the missing `batchEnabled` check (see H-01) is implemented.

Impact: Informational, but affects overall token control and lifecycle.

Mitigation: Introduce a deadline or expiration mechanism per batch (e.g., using a `batchExpiry` mapping), and integrate it into the `claim` logic.

RESOLVED

[I-03] Missing Zero Address Check

Description: There is no check to ensure that `signer, vault` is not the zero address. This could lead to unintended behavior or misuse,

Impact: Informational, but recommended for robust contract safety.

Mitigation: Add a validation check like:

```
if (signer == address(0)) {
    revert ZeroAddress();
}
```

RESOLVED

.

About SAFE EDGES

SafeEdges is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With six years of expertise, we've secured over 1000 projects globally, averting over $30 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

500+
Audits Completed

$3B
Secured

600k+
Lines of Code Audited