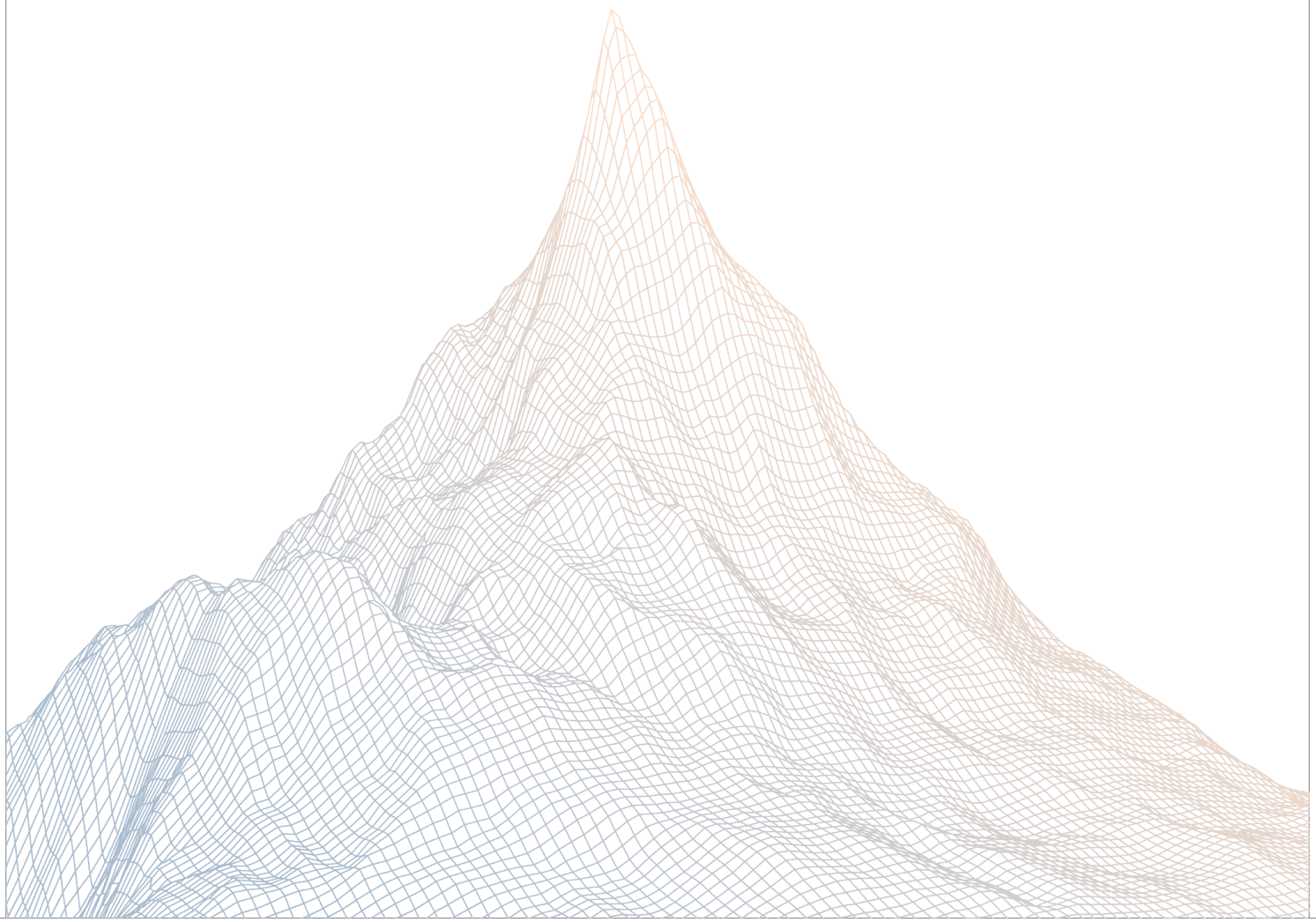
Zenith

Spiko

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: November 4th to November 5th, 2025

AUDITED BY: Arno
valuevalk

Contents

1 Introduction

1.1 About Zenith

Zenith assembles auditors with proven track records: finding critical vulnerabilities in public audit competitions.

Our audits are carried out by a curated team of the industry's top-performing security researchers, selected for your specific codebase, security needs, and budget.

Learn more about us at https://zenith.security.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2

Executive Summary

2.1 About Spiko

At Spiko, we believe financial services infrastructure should be digital, open and composable.

We are leveraging distributed technology to build the next generation open-source securities issuance, management and distribution platform.

2.2 Scope

The engagement involved a review of the following targets:

Target	contracts
Repository	https://github.com/spiko-tech/contracts
Commit Hash	`2ae1d91c07977206b62a76c28c2b44b8d658007c`
Files	`contracts/token/Minter.sol`

2.3 Audit Timeline

November 4, 2025	Audit start
November 5, 2025	Audit end
November 7, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	2
Informational	3
Total Issues	**5**

3

Findings Summary

ID	Description	Status
L-1	Expired status in mintStateStatus doesn't account for deadline's boundary	Acknowledged
L-2	initiateMint is callable even when daily limit and maxDelay fields are not set yet	Acknowledged
I-1	approveMint doesn't consume daily quota even if its available	Acknowledged
I-2	initiateMint, approveMint, cancelMint lack basic validations for input fields	Resolved
I-3	No Initializer for Upgradeable Contract	Acknowledged

4

Findings

4.1 Low Risk

A total of 2 low risk findings were identified.

[L-1] Expired status in mintStateStatus doesn't account for deadline's boundary

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Medium

Target

- Minter.sol#L49-L60

Description:

The `Expired` status in mintStateStatus doesn't account for deadline's boundary:

```
    function mintStateStatus(bytes32 id) public view returns (Status) {
        uint256 deadline = mintStateDeadline(id);
        if (deadline == 0) {
            return Status.NULL;
        } else if (deadline == type(uint256).max) {
            return Status.DONE;
        } else if (deadline > block.timestamp) {
            return Status.PENDING;
        } else {
@>>         //if deadline is == block.timestamp we are still expired
            return Status.EXPIRED;
        }
    }
```

Possible flow:

1. Daily limit is surpassed, so a `mint operation` is moved to `PENDING` state by adding it a specific deadline via the `initiateMint()` flow
2. `approveMint()` cannot be called on the boundary block because when `deadline == block.timestamp` the state moves to EXPIRED

Result is: Deadline expires 1 block earlier (seconds differ per chain slot finality time), because the deadline block is not inclusive.

Recommendations:

To account for the full duration, including for the deadline's boundary as well:

```
function mintStateStatus(bytes32 id) public view returns (Status) {
    uint256 deadline = mintStateDeadline(id);
    if (deadline == 0) {
        return Status.NULL;
    } else if (deadline == type(uint256).max) {
        return Status.DONE;
    } else if (deadline > block.timestamp) {
    } else if (deadline ≥ block.timestamp) {
        return Status.PENDING;
    } else {
        return Status.EXPIRED;
    }
}
```

Spiko: Acknowledged. Working as Intended. This behavior is intentional. Our deadline is designed to be non-inclusive by design—operations must complete before the deadline timestamp, not at it.

- Consistent with our protocol's timing semantics.
- The 1-block difference creates no functional or security issues.
- Our operations and monitoring account for this strict boundary.

Status: Will Not Fix - Design decision.

[L-2] initiateMint is callable even when daily limit and maxDelay fields are not set yet

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- Minter.sol#L94-L114

Description:

The `Minter.sol` contract is created without setting `dailyLimit` and `_maxDelay` as there is no initialize function.

This allows for `initiateMint` to be called before that, which will either:

1. Immediately move the initiated mint into `EXPIRED` state (if no maxDelay is set)
2. Or will just immediately move it to pending (if only `dailyLimit` for that particular token is not set), because `currentUsage + amount` will always be over 0, for normal input field amounts.

```
@>>  function initiateMint(address user, Token token, uint256 amount,
    bytes32 salt) external restricted {
        DailyUsage storage tokenDailyUsage = _dailyUsage[token];
        bytes32 id = hashMintId(user, token, amount, salt);

        require(mintStateStatus(id) == Status.NULL, "ID already used");

        uint256 currentDay = getCurrentDay();
        uint256 currentUsage = getMintedToday(token);

@>>         if (currentUsage + amount > tokenDailyUsage.dailyLimit) {
            _mintDeadline[id] = block.timestamp + _maxDelay;
            emit MintBlocked(id, user, token, amount, salt);
@>>         } else {
            tokenDailyUsage.lastUsageDay = currentDay;
            tokenDailyUsage.dailyUsed = currentUsage + amount;
            _mintDeadline[id] = type(uint256).max;
            token.mint(user, amount);
```

```
            emit MintExecuted(id, user, token, amount, salt);
        }
    }
```

Recommendations:

For explicit safety, you can add the following validations:

```
function initiateMint(address user, Token token, uint256 amount,
    bytes32 salt) external restricted {
      DailyUsage storage tokenDailyUsage = _dailyUsage[token];
      bytes32 id = hashMintId(user, token, amount, salt);

      require(mintStateStatus(id) == Status.NULL, "ID already used");
    require(_maxDelay ≠ 0, "maxDelay is not set, yet");
    require(tokenDailyUsage ≠ 0, "tokenDailyUsage is not set, yet");

      uint256 currentDay = getCurrentDay();
      uint256 currentUsage = getMintedToday(token);

     if (currentUsage + amount > tokenDailyUsage.dailyLimit) {
          _mintDeadline[id] = block.timestamp + _maxDelay;
          emit MintBlocked(id, user, token, amount, salt);
      } else {
          tokenDailyUsage.lastUsageDay = currentDay;
          tokenDailyUsage.dailyUsed = currentUsage + amount;
          _mintDeadline[id] = type(uint256).max;
          token.mint(user, amount);
          emit MintExecuted(id, user, token, amount, salt);
      }
  }
```

Spiko: Acknowledged. Working as Intended. The absence of an initialize function and the ability to call initiateMint before setting dailyLimit and _maxDelay is intentional. These parameters are designed to be configured post-deployment as part of our operational setup process.

- Our deployment workflow explicitly sets these parameters after contract creation
- Controlled deployment environment prevents unauthorized calls before configuration

Status: Will Not Fix - Intentional deployment sequence.

4.2 Informational

A total of 3 informational findings were identified.

[I-1] approveMint doesn't consume daily quota even if its available

SEVERITY: Informational	IMPACT: Informational
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- Minter.sol#L119-L128

Description:

As described in the readme there are 2 flow paths:

> When an authorized operator initiates a mint via initiateMint():
>
> Immediate Execution: If the mint amount would not exceed the daily limit (i.e., currentUsage + amount <= dailyLimit), the tokens are minted immediately to the user
>
> Pending State: If the mint amount would exceed the daily limit, the operation is placed in a pending state with a deadline set to block.timestamp + maxDelay The deadline can be configured by administrators via setMaxDelay()
>
> Pending operations can be in one of four states: NULL (doesn't exist), PENDING (waiting for approval), EXPIRED (deadline passed), or DONE (executed or canceled)

Regarding the pending state path its also mentioned:

> The Minter contract ensures that if a mint operation would exceed the daily limit, the operation is deferred to a pending state that can later be reviewed and either approved or canceled by authorized operators. This design allows Spiko to handle high subscription volumes while maintaining control over daily issuance rates.

There is a case where:

1. `initiateMint` is called on Day 1, where limit is already surpassed and mint with id `17` is moved to pending state
2. However, this can happen at any time, including close to the end of Day 1.
3. Let's assume that 30 mins later, its already Day 2.
4. Quota, i.e. dailyLimit has been reset, so there is room for mint `17` already
5. However now when the mint is approved on Day 2 and the mint itself is executed via `approveMint`, the `amount` minted isn't accounted for in the `tokenDailyUsage.dailyUsed` even though there is an available quota.

Code reference, approveMint doesn't account for the quota.

```
function approveMint(address user, Token token, uint256 amount,
    bytes32 salt) external restricted {
    bytes32 id = hashMintId(user, token, amount, salt);

    require(mintStateStatus(id) == Status.PENDING, "Operation is not
    pending");
    _mintDeadline[id] = type(uint256).max; // Mark as executed

    token.mint(user, amount);

    emit MintExecuted(id, user, token, amount, salt);
}
```

The logic works as of now as well, however it should be acknowledged that its fully agnostic of the possible available quota.

Recommendations

If you prefer to keep `approveMint` fully agnostic of the dailyLimit, after the initiation was already done when the quota was surpassed -> **no change is needed**.

If you would like to tie the `approveMint` to consuming any available quota, if for example we've switched to the next day and the limit is reset, enforce that into the `approveMint` function by updating the `dailyUsed` for the specific token, only when `currentUsage + amount ≤ tokenDailyUsage.dailyLimit`, similarly to how its done in `initiateMint()`

Spiko: Acknowledged. Working as Intended. The `approveMint` function is intentionally designed to be fully agnostic of daily limits and quota tracking.- `approveMint` serves a single purpose: approve or reject a pending operation

- Once a mint enters the pending state, it has already been reviewed against the daily limit during initiateMint

- The approval process is a manual override mechanism that bypasses quota enforcement by design
- Daily limit tracking is exclusively handled in initiateMint where quota management belongs

Status: Will Not Fix - Intentional separation of concerns.

[I-2] initiateMint, approveMint, cancelMint lack basic validations for input fields

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- Minter.sol#L94-L114

Description:

`initiateMint()`, `cancelMint`, `approveMint` functions currently lacks input validation for the amount and token address, adding them can make the call fail earlier and more explicitly. (you can also add custom errors if you prefer to)

```
@>>     function initiateMint(address user, Token token, uint256 amount,
    bytes32 salt) external restricted {
```

Recommendations:

To avoid basic pitfalls you can add the following basic checks:

```
function initiateMint(address user, Token token, uint256 amount,
    bytes32 salt) external restricted {
    require(amount > 0, "the mint amount can't be 0");
    require(address(token) ≠ address(0), "token address must be valid");
    DailyUsage storage tokenDailyUsage = _dailyUsage[token];
    bytes32 id = hashMintId(user, token, amount, salt);

    require(mintStateStatus(id) == Status.NULL, "ID already used");
```

Consider adding the same validations to the other funcs.

Spiko: Resolved with `@c0b1a75747 ...`. Input validation for amount and token address has been added to `initiateMint()`, `cancelMint()`, and `approveMint()` functions as recommended.

Zenith: Verified.

[I-3] No Initializer for Upgradeable Contract

SEVERITY: Informational	IMPACT: Informational
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- Minter.sol

Description:

The contract inherits from `UUPSUpgradeable` but lacks an initializer function, leaving storage variables like `_maxDelay` at its default value of 0. This renders the contract non-functional for pending mints until an admin calls `setMaxDelay()`, potentially causing delayed usability.

Recommendations:

Add an initializer function to set _maxDelay and other necessary variables during proxy deployment:

```
// Add to contract
function initialize(uint256 initialMaxDelay) external initializer {
    _maxDelay = initialMaxDelay;
    // Add other initializations if needed
}
```

Spiko: Acknowledged. Working as Intended. The absence of an initializer function is intentional. Our deployment process explicitly configures _maxDelay and other parameters post-deployment via admin functions.

- Deployment workflow is controlled and deliberate
- Admin functions like `setMaxDelay()` provide the necessary configuration mechanism
- No operational issues arise from this approach given our controlled deployment environment

Status: Will Not Fix - Intentional configuration approach.