ERC1155 Eligibility Minter V1 Review

September 8, 2025

Prepared for ZKsync

Conducted by:

Richie Humphrey (devtooligan)

About the ZKsync ZkMinterERC1155EligibilityV1 Review

ZKsync Era is a Layer 2 ZK rollup that uses cryptographic validity proofs to provide scalable and low-cost transactions on Ethereum. The `ZkMinterERC1155EligibilityV1` contract extends the capped minter functionality by implementing ERC1155-based eligibility checks and rate limiting for token distribution. This system allows controlled token minting where eligibility is determined by ERC1155 token holdings, providing a throttled mechanism for streaming tokens from the capped minter.

About Offbeat Security

Offbeat Security is a boutique security company providing unique security solutions for complex and novel crypto projects. Our mission is to elevate the blockchain security landscape through invention and collaboration.

Summary & Scope

The src folder was reviewed at commit aa60218.

The following **2 files** were in scope:

- src/ZkMinterERC1155EligibilityV1.sol
- src/ZkMinterERC1155EligibilityV1Factory.sol

The protocol implements gated minting functionality using ERC1155 token holdings to determine eligibility. The factory pattern enables deterministic deployment of minter instances with configurable parameters for balance thresholds and token IDs.

Summary of Findings

Identifier	Title	Severity	Fixed
L-01	Missing ERC165 interface check for ERC1155 contract in constructor	Low	Fixed in PR#35
L-02	Bytecode hash mismatch can cause incorrect address prediction	Low	Ack. This issue was also surfaced on the review of the Capped Minter. Due to challenges related to the compiler, the project has noted they will not fix this issue and will use extra care around deployments.

Detailed Findings

Low Findings

[L-01] Missing ERC165 interface check for ERC1155 contract in constructor

Description

The `ZkMinterERC1155EligibilityV1` contract accepts an ERC1155 contract address in its constructor but does not verify that the provided address actually implements the

ERC1155 interface. The constructor directly casts the address to `IERC1155` without validation:

```
ERC1155 = IERC1155(_erc1155);
```

If an incorrect address (such as an EOA or a contract that doesn't implement ERC1155) is provided during deployment, the contract will be deployed successfully but will fail when attempting to check balances through `ERC1155.balanceOf()` in the `_isEligible()` function. This results in a non-functional contract that would need to be redeployed.

According to the ERC1155 standard, compliant contracts must implement ERC165's `supportsInterface()` function with the ERC1155 interface ID (`0xd9b67a26`). The contract should verify this support during construction to prevent deployment with invalid contracts.

Recommendation

Consider adding an ERC165 interface check in the constructor to verify that the provided address implements the ERC1155 interface:

```
+ import {IERC165} from "@openzeppelin/contracts/utils/introspection/IERC165.sol'

contract ZkMinterERC1155EligibilityV1 is ZkMinterV1 {

    constructor(IMintable _mintable, address _admin, address _erc1155, uint256 _
        if (_admin == address(0)) {
            revert ZkMinterERC1155EligibilityV1__InvalidZeroAddress();
        }
+
+       // Verify the provided address implements ERC1155
+       if (!IERC165(_erc1155).supportsInterface(type(IERC1155).interfaceId)) {
+           revert ZkMinterERC1155EligibilityV1__InvalidERC1155Contract();
+       }

        ERC1155 = IERC1155(_erc1155);
        _updateMintable(_mintable);
        _updateTokenId(_tokenId);
        _updateBalanceThreshold(_balanceThreshold);

        _grantRole(DEFAULT_ADMIN_ROLE, _admin);
        _grantRole(PAUSER_ROLE, _admin);
    }
```

[L-02] Bytecode hash mismatch can cause incorrect address prediction

Description

The `ZkMinterERC1155EligibilityV1Factory` contract accepts a bytecode hash as a constructor parameter without verifying it matches the actual deployment bytecode of `ZkMinterERC1155EligibilityV1`. This could lead to incorrect address predictions by the `getMinter()` function if the provided hash doesn't match the actual deployed contract bytecode.

The factory constructor accepts a `_bytecodeHash` parameter that is stored as an immutable variable:

```
constructor(bytes32 _bytecodeHash) {
    BYTECODE_HASH = _bytecodeHash;
}
```

This hash is used in `getMinter()` to predict deployment addresses:

```
function getMinter(
    IMintable _mintable,
    address _admin,
    address _erc1155,
    uint256 _tokenId,
    uint256 _balanceThreshold,
    uint256 _saltNonce
) external view returns (address _minterERC1155Address) {
    bytes32 _salt = _calculateSalt(_saltNonce);
    _minterERC1155Address = L2ContractHelper.computeCreate2Address(
        address(this),
        _salt,
        BYTECODE_HASH,
        keccak256(abi.encode(_mintable, _admin, _erc1155, _tokenId, _balanceThre
    );
}
```

Two potential issues arise:

1. The constructor parameter could be set incorrectly during deployment
2. Small changes in the contract's metadata (comments, whitespace, or unused files) can change the bytecode hash, making it diverge from the provided value

If either occurs, `getMinter()` will return incorrect addresses, potentially causing issues for users and integrating protocols that rely on address prediction.

Recommendation

Consider calculating the bytecode hash during construction instead of accepting it as a parameter. One approach could be to use the `hashL2Bytecode` function from the ZKsync Utils library to calculate the hash at deployment time.

This change would ensure the bytecode hash always matches the actual deployment code and eliminates the possibility of configuration errors. It also makes the contract more maintainable as it automatically updates when the contract code changes.