Evo SoulBoundToken Audit Report

Prepared by Cyfrin

Version 2.0

Lead Auditors

Dacian

Giovanni Di Siena

June 2, 2025

Contents

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

Evo Labs has created the `SoulBoundToken` contract as a non-transferable NFT to be used for voting within their DAO. The contract will be deployed on Optimism and allows minting only one non-transferable NFT per address via the following functions:

- `mintAsAdmin`, `batchMintAsAdmin` - only callable by admins, cost nothing to mint
- `mintAsWhitelisted` - only callable by whitelisted addresses who must pay the mint fee
- `mintWithTerms` - callable buy any public address who has signed the Terms Of Service and must also pay a fee

While the contract is immutable, it has two privileged roles: one owner and one or more admins. The owner has access to the following functions:

- `setAdmin`, `batchSetAdmin` - enables/disables admin status of input addresses
- `withdrawFees` - collects the NFT minting fees
- `setContractURI` - updates the Terms Of Service reference

Admins have access to the following non-minting functions:

- `addToWhitelist`, `batchAddToWhitelist` - adds addresses to whitelist
- `removeFromWhitelist`, `batchRemoveFromWhitelist` - remove addresses from whitelist
- `addToBlacklist`, `batchAddToBlacklist` - adds addresses to blacklist and if they own an NFT, burns it
- `setWhitelistEnabled` - toggle whitelist on/off
- `setFeeFactor` - changes the NFT mint cost which is calculated based on the price of Ethereum using a Chainlink price feed

Though the NFT is designed to be used for voting in a DAO, the `SoulBoundToken` contract does not itself contain any voting-related functionality or code; its only external integration is with the Chainlink price feed used to calculate the mint fee.

5 Audit Scope

The only file in scope was `src/SoulBoundToken.sol`.

6 Executive Summary

Over the course of 2 days, the Cyfrin team conducted an audit on the Evo SoulBoundToken smart contracts provided by Evo Labs. In this period, a total of 15 issues were found.

The findings consist of 5 Low severity issues with the rest being gas optimizations and informational.

Code & Test Suite Analysis

The code has been written very defensively and professionally to a very high standard. Similarly the test suite contains unit tests, invariant fuzz tests and certora formal verification specifications. The development team has really done an amazing job of hardening their code as much as possible prior to external audit and should be commended.

Summary

Project Name	Evo SoulBoundToken
Repository	sbt
Commit	124f8c621de9...
Audit Timeline	May 29th - May 30th, 2025
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	5
Informational	6
Gas Optimizations	4
Total Issues	15

Summary of Findings

[L-1] Missing common Chainlink Oracle validations	Resolved
[L-2] Users who are removed from the blacklist have to pay again for their NFT	Acknowledged
[L-3] Round up fee against users	Resolved
[L-4] Whale can buy near-infinite voting power via `mintWithTerms`	Resolved

[L-5] `_verifySignature` is not compatible with smart contract wallets or other smart accounts	Resolved
[I-1] Prefer`Ownable2Step` instead of `Ownable`	Resolved
[I-2] Assuming Chainlink price feed decimals can lead to unintended errors	Resolved
[I-3] Don't initialize to default values	Resolved
[I-4] Remove obsolete `return` statements when already using named returns	Resolved
[I-5] Consider preventing users from over-paying	Resolved
[I-6] `else` can be omitted in `mintWithTerms`	Resolved
[G-1] Use named returns where this can eliminate local function variables and for `memory` returns	Resolved
[G-2] Enable the optimizer	Resolved
[G-3] Prefer `calldata` to `memory` for external read-only inputs	Resolved
[G-4] Use solady `safeTransferETH` to send eth	Resolved

7 Findings

7.1 Low Risk

7.1.1 Missing common Chainlink Oracle validations

Description: The protocol is missing common Chainlink Oracle validations; it calls 'AggregatorV3Interface::latestRoundData' without any validation of the result:

```
function _getLatestPrice() internal view returns (uint256) {
    //slither-disable-next-line unused-return
    (, int256 price,,,) = i_nativeUsdFeed.latestRoundData();
    return uint256(price);
}
```

Recommended Mitigation: Implement common Chainlink oracle validations such as checking for:

- stale prices using the correct heartbeat for the particular oracle
- down L2 sequencer, revert if `startedAt == 0` and potentially a small grace period of ~2 minutes after it recovers before resuming to fetch price data
- returned price not at min or max boundaries

For this protocol the impact of omitting these checks is quite minimal; in a worst-case scenario users are able to buy NFTs for a cheaper or greater price, but there is no threat to protocol solvency/user liquidation etc as can be a threat in other protocols. And since users can only buy 1 NFT and can't sell/transfer, it isn't that big a deal. If these checks are excluded to keep gas costs down perhaps just put a comment noting this.

Evo: Fixed in commits 6af531e,7a06688, 93021e4.

Cyfrin: Verified.

7.1.2 Users who are removed from the blacklist have to pay again for their NFT

Description: When a user is added to the blacklist, the NFT which they already paid for is burned:

```
function _addToBlacklist(address account) internal {
    // *snip: code not relevant *//

    if (balanceOf(account) > 0) {
        uint256 tokenId = tokenOfOwnerByIndex(account, 0); // Get first token
        _burn(tokenId); // Burn the token
    }
}
```

But when they are removed from the blacklist, they do not receive a free NFT to make up for their previously burned one, nor is there any flag set that would enable them to mind their NFT again but without paying a fee:

```
function _removeFromBlacklist(address account) internal {
    if (!s_blacklist[account]) revert SoulBoundToken__NotBlacklisted(account);

    s_blacklist[account] = false;
    emit RemovedFromBlacklist(account);
}
```

Impact: A user who bought an NFT, then was blacklisted, then removed from the blacklist will have to pay twice to get the NFT.

Recommended Mitigation: This doesn't seem fair; if a user had an NFT burned when they were blacklisted, they should receive a free NFT back if later removed from the blacklist.

Evo: Acknowledged; in the unlikely case a user is blacklisted due to admin error then subsequently removed from the blacklist, the DAO will compensate the user via a community vote.

7.1.3 Round up fee against users

Description: Solidity by default rounds down, but generally fees should be rounded up against users. Using Solady's library is significantly more efficient than OpenZeppelin:

```
import {FixedPointMathLib} from "@solady/utils/FixedPointMathLib.sol";

function _getFee() internal view returns (uint256 fee) {
    // read fee factor directly to output variable
    fee = s_feeFactor;

    // only do extra work if non-zero
    if(fee != 0) fee = FixedPointMathLib.fullMulDivUp(fee, PRICE_FEED_PRECISION, _getLatestPrice());
}
```

A secondary benefit of using the above is eliminating the possibility of revert due to intermediate multiplication overflow, though in this code it isn't a real possibility.

If you don't want to round up against users but want a slightly faster implementation than the default:

```
function _getFee() internal view returns (uint256 fee) {
    // read fee factor directly to output variable
    fee = s_feeFactor;

    // only do extra work if non-zero
    if(fee != 0) fee = (fee * PRICE_FEED_PRECISION) / _getLatestPrice();
}
```

Evo: Fixed in commit 52c5384.

Cyfrin: Verified.

7.1.4 Whale can buy near-infinite voting power via `mintWithTerms`

Description: Since whales can control near-infinite addresses, as long as they have the funds they can buy near-infinite voting power via `mintWithTerms`.

Impact: This could be used seconds before a proposal is due to expire to decide that proposal. Long-term impact is limited however since the admins can blacklist addresses which burn the NFTs.

Recommended Mitigation: Implement a snapshot mechanism to capture total and individual user voting power prior to proposals. Consider implementing pausing to prevent users from minting NFTs via `mintWithTerms` since this is the only function which allows "unlimited mints".

Evo: Fixed in commit 2fbd2c5 by allowing admins to pause `mintWithTerms`.

Cyfrin: Verified.

7.1.5 `_verifySignature` is not compatible with smart contract wallets or other smart accounts

Description: Support for smart accounts (e.g. ERC-4337) and other smart contract wallets (e.g. Safe{Wallet}) minting tokens is not currently possible as the signature verification implemented in `_verifySignature` is only able to handle those generated by EOAs. Here, it could be beneficial to support signature verification not just for smart accounts but also other smart contracts that could include multi-sig wallets or any other use case, for example DAOs with their own smart contract infrastructure, to allow other organizations to participate as members.

EOAs upgraded to ERC-7702 accounts are unaffected, but any other smart contract signatures cannot be verified without implementing ERC-1271. However, this adds the additional consideration that for EIP-7702 accounts the code length will be non-zero, so while these accounts can have their signatures verified using EIP-1271, the private

key still holds full authority to sign transactions which means that any implementation of a code length check such as in the OpenZeppelin SignatureChecker library will need to be slightly modified to continue to allow verification of signatures from these accounts generated using `eth/personal_sign`. Additional discussion can be found here.

To support such smart contract signatures, consider falling back to the OpenZeppelin SignatureChecker library function `isValidERC1271SignatureNow` like so:

```
    function _verifySignature(bytes memory signature) internal view returns (bool) {
        /// @dev compute the message hash: keccak256(termsHash, msg.sender)
        bytes32 messageHash = keccak256(abi.encodePacked(s_termsHash, msg.sender));

        /// @dev apply Ethereum signed message prefix
        bytes32 ethSignedMessageHash = MessageHashUtils.toEthSignedMessageHash(messageHash);

        /// @dev attempt to recover the signer
        //slither-disable-next-line unused-return
        (address recovered, ECDSA.RecoverError error,) = ECDSA.tryRecover(ethSignedMessageHash,
        ↪  signature);

        /// @dev return false if errors or incorrect signer
++      if (error == ECDSA.RecoverError.NoError && recovered == msg.sender) return true;
++      else return SignatureChecker.isValidERC1271SignatureNow(msg.sender, ethSignedMessageHash,
↪  signature);
    }
```

Evo: Fixed in commit 6d4f41c.

Cyfrin: Verified.

7.2 Informational

7.2.1 Prefer`Ownable2Step` instead of `Ownable`

Description: Prefer Ownable2Step instead of `Ownable` for safer ownership transfer.

Evo: Fixed in commit 620120e.

Cyfrin: Verified.

7.2.2 Assuming Chainlink price feed decimals can lead to unintended errors

Description: In general, Chainlink x/USD price feeds use 8 decimal precision however this is not universally true for example AMPL/USD uses 18 decimal precision.

Instead of assuming Chainlink oracle price precision, the precision variable could be declared `immutable` and initialized in the constructor via `AggregatorV3Interface::decimals`.

In practice though the price oracle is hard-coded in `script/HelperConfig.s.sol` and does use 8 decimals for on Optimism, so the current configuration will work fine.

Evo: Fixed in commit f594ae0.

Cyfrin: Verified.

7.2.3 Don't initialize to default values

Description: Don't initialize to default values as Solidity already does this:

```
SoulBoundToken.sol
125:         for (uint256 i = 0; i < admins.length; ++i) {
162:         for (uint256 i = 0; i < accounts.length; ++i) {
226:         for (uint256 i = 0; i < accounts.length; ++i) {
245:         for (uint256 i = 0; i < accounts.length; ++i) {
270:         for (uint256 i = 0; i < accounts.length; ++i) {
289:         for (uint256 i = 0; i < accounts.length; ++i) {
312:         for (uint256 i = 0; i < accounts.length; ++i) {
```

Evo: Fixed in commit f594ae0.

Cyfrin: Verified.

7.2.4 Remove obsolete `return` statements when already using named returns

Description: Remove obsolete `return` statements when already using named returns:

```
    function _mintSoulBoundToken(address account) internal returns (uint256 tokenId) {
        tokenId = _incrementTokenIdCounter(1);
        _safeMint(account, tokenId);
-       return tokenId;
    }

    function _incrementTokenIdCounter(uint256 count) internal returns (uint256 startId) {
        startId = s_tokenIdCounter;
        s_tokenIdCounter += count;
-       return startId;
    }
```

Evo: Fixed in commit f594ae0.

Cyfrin: Verified.

7.2.5 Consider preventing users from over-paying

Description: Currently the protocol allows users to over-pay:

```
function _revertIfInsufficientFee() internal view {
    if (msg.value < _getFee()) revert SoulBoundToken__InsufficientFee();
}
```

Consider changing this to require the exact fee to prevent users from accidentally over-paying:

```
function _revertIfIncorrectFee() internal view {
    if (msg.value != _getFee()) revert SoulBoundToken__IncorrectFee();
}
```

Fat Finger errors have previously resulted in notorious unintended errors in financial markets; the protocol could choose to be defensive and help protect users from themselves.

Evo: Fixed in commit e3b2f74.

Cyfrin: Verified.

7.2.6 `else` can be omitted in `mintWithTerms`

Description: `else` can be omitted in `mintWithTerms` since if signature validation failed a revert will occur:

```
        if (!_verifySignature(signature)) revert SoulBoundToken__InvalidSignature();
-       else emit SignatureVerified(msg.sender, signature);
+       emit SignatureVerified(msg.sender, signature);
        tokenId = _mintSoulBoundToken(msg.sender);
```

Evo: Fixed in commit e3b2f74.

Cyfrin: Verified.

7.3 Gas Optimization

7.3.1 Use named returns where this can eliminate local function variables and for `memory` returns

Description: Using named returns is more gas efficient where this can eliminate local function variables and for `memory` returns:

```
-   function batchMintAsAdmin(address[] calldata accounts) external onlyAdmin returns (uint256[]
↪   memory) {
+   function batchMintAsAdmin(address[] calldata accounts) external onlyAdmin returns (uint256[] memory
↪   tokenIds) {
        _revertIfEmptyArray(accounts);
        uint256 startId = _incrementTokenIdCounter(accounts.length);

-       uint256[] memory tokenIds = new uint256[](accounts.length);
+       tokenIds = new uint256[](accounts.length);
        for (uint256 i = 0; i < accounts.length; ++i) {
            _mintAsAdminChecks(accounts[i]);
            tokenIds[i] = startId + i;
            _safeMint(accounts[i], tokenIds[i]);
        }
-       return tokenIds;
    }
```

Gas Result:

```
{
-   "batchMintAsAdmin": "252114"
+   "batchMintAsAdmin": "252102"
}
```

Evo: Fixed in commit b4fcadb.

Cyfrin: Verified.

7.3.2 Enable the optimizer

Description: Enable the optimizer in `foundry.toml`.

Gas results:

```
{
-   "addToBlacklist": "31090"
+   "addToBlacklist": "30691"

-   "addToWhitelist": "28754"
+   "addToWhitelist": "28392"

-   "batchAddToBlacklist": "60282"
+   "batchAddToBlacklist": "59482"

-   "batchAddToWhitelist": "55790"
+   "batchAddToWhitelist": "54997"

-   "batchMintAsAdmin": "252102"
+   "batchMintAsAdmin": "248867"

-   "batchRemoveFromBlacklist": "5289"
+   "batchRemoveFromBlacklist": "4594"

-   "batchRemoveFromWhitelist": "5305"
+   "batchRemoveFromWhitelist": "4677"
```

```
-  "batchSetAdmin": "28090"
+  "batchSetAdmin": "27412"

-  "mintAsAdmin": "130754"
+  "mintAsAdmin": "129447"

-  "mintAsWhitelisted": "135623"
+  "mintAsWhitelisted": "132292"

-  "mintWithTerms": "142281"
+  "mintWithTerms": "137638"

-  "removeFromBlacklist": "2516"
+  "removeFromBlacklist": "2203"

-  "removeFromWhitelist": "2634"
+  "removeFromWhitelist": "2254"

-  "setAdmin": "27187"
+  "setAdmin": "26677"

-  "setContractURI": "29118"
+  "setContractURI": "26842"

-  "setFeeFactor": "26075"
+  "setFeeFactor": "25666"

-  "setWhitelistEnabled": "7175"
+  "setWhitelistEnabled": "6902"

-  "withdrawFees": "14114"
+  "withdrawFees": "13462"
}
```

Evo: Fixed in commit b4fcadb.

Cyfrin: Verified.

7.3.3 Prefer `calldata` to `memory` for external read-only inputs

Description: Prefer `calldata` to `memory` for external read-only inputs:

```
-   function mintWithTerms(bytes memory signature) external payable returns (uint256 tokenId) {
+   function mintWithTerms(bytes calldata signature) external payable returns (uint256 tokenId) {

-   function _verifySignature(bytes memory signature) internal view returns (bool) {
+   function _verifySignature(bytes calldata signature) internal view returns (bool) {
```

Gas results:

```
{
-  "mintWithTerms": "137638"
+  "mintWithTerms": "137299"
}
```

Evo: Fixed in commit b4fcadb.

Cyfrin: Verified.

7.3.4 Use solady `safeTransferETH` to send eth

Description: Using solady `safeTransferETH` is a more efficient way to send eth. Also since there is no point in leaving eth inside the contract, consider removing the `amountToWithdraw` input parameter and checks associated with it; instead just send the entire contract balance:

```
function withdrawFees() external onlyOwner {
    uint256 amountToWithdraw = address(this).balance;
    if(amountToWithdraw > 0) {
        // from https://github.com/Vectorized/solady/blob/main/src/utils/SafeTransferLib.sol#L90-L98
        /// @solidity memory-safe-assembly
        assembly {
            if iszero(call(gas(), caller(), amountToWithdraw, codesize(), 0x00, codesize(), 0x00)) {
                mstore(0x00, 0xefde920d) // `SoulBoundToken__WithdrawFailed()`.
                revert(0x1c, 0x04)
            }
        }

        emit FeesWithdrawn(amountToWithdraw);
    }
}
```

Gas result:

```
{
- "withdrawFees": "13462"
+ "withdrawFees": "13353"
}
```

Evo: Fixed in commit b4fcadb.

Cyfrin: Verified.