OpenZeppelin | security

# SolvBTC Blacklist Audit

December 27, 2024

# Table of Contents

# Summary

| | |
|---|---|
| Type | DeFi |
| Timeline | From 2024-12-12<br>To 2024-12-13 |
| Languages | Solidity |

| | |
|---|---|
| Total Issues | 16 (16 resolved) |
| Critical Severity Issues | 0 (0 resolved) |
| High Severity Issues | 1 (1 resolved) |
| Medium Severity Issues | 0 (0 resolved) |
| Low Severity Issues | 2 (2 resolved) |
| Notes & Additional Information | 13 (13 resolved) |

# Scope

We audited the solv-finance/SolvBTC repository at commit 044f73b.

In scope were the following files:

```
contracts
├── ISolvBTC.sol
├── SolvBTCV2_1.sol
├── SolvBTCV3.sol
└── access
    └── BlacklistableUpgradeable.sol
```

## System Overview

SolvBTC is a yield-bearing, upgradeable ERC-20 token designed to integrate with multiple blockchain networks such as Ethereum, Arbitrum, BNB Chain, and Merlin Chain. The Solv Protocol serves as a unified liquidity gateway, connecting these different networks to create a more inclusive BTCFi ecosystem. The contracts under review implement SolvBTC with an added blacklist feature for greater control and security. The contracts are ownable, meaning they are managed by an owner who can oversee and administer critical operations.

The blacklist is governed by a blacklist manager set by the owner. Once an address is blacklisted, it cannot interact with the token. This restriction includes transferring tokens or using previously approved allowances, ensuring that blacklisted addresses cannot move or access their funds. The token also includes mint and burn functions controlled by three roles. Addresses with the `SOLVBTC_MINTER_ROLE` can mint tokens to any account and burn tokens from their own balance. Addresses with the `SOLVBTC_POOL_BURNER_ROLE` can burn tokens held by any account. Finally, the owner of the contract is able to burn tokens that are held by blacklisted addresses.

## Trust Assumptions and Privileged roles

During the audit, the following trust assumptions were made about the various privileged roles:

- The `blacklistManager` is trusted not to abuse the blacklist functionality. It is assumed that external communications will be made regarding which rules define who is placed on the blacklist, and that the owner will adhere to these.
- The `SOLVBTC_POOL_BURNER_ROLE` is trusted to burn tokens from other accounts only when necessary, as defined by the Solv protocol.
- The `SOLVBTC_MINTER_ROLE` is trusted to only mint tokens as defined by the Solv protocol. It is also trusted to promptly burn tokens which it holds as defined by the Solv protocol.

The `owner` is trusted to monitor and manage the holders of all other roles, ensuring that no malicious entities retain control of these roles in the event that an account is compromised or an admin goes rogue. The contract owner is trusted to have control of the predefined address within the `initialize` function of the `SolvBTCV2_1` contract.

All privileged roles are trusted to keep any necessary private keys safely and to engage in good operational security to reduce the chances of phishing attacks or engaging with malware. All privileged roles are trusted to monitor their accounts for suspicious activity. In addition, privileged roles should have a plan in place for quickly transferring roles to a fresh account that they control if malicious behavior is detected, and then to notify users of this change.

When blacklisting accounts, the `blacklistManager` is trusted to utilize Flashbots or some other private mempool service to broadcast the blacklist transactions. This will avoid front-running, whereby users see an incoming blacklist transaction in the mempool, and move their funds to fresh accounts to evade it.

# High Severity

## H-01 Blacklisted Users Can Spend Allowances

If a user is blacklisted, they are not prevented from transferring tokens possessed by another account. This occurs because the `_update` function only checks whether the `from` and `to` addresses are on the blacklist. As a result, a blacklisted user with existing approvals for another account's tokens can still spend those tokens.

Note that the `_approve` function has a check on `spender` which prevents blacklisted users from obtaining new approvals. However, this does not mitigate the risk of using previously granted approvals.

Consider adding a check to the `_update` function which ensures that `msg.sender` is not blacklisted.

***Update:** Resolved in pull request #8 at commit 5eb5140. The team stated:*

> *Modify according to the suggestions.*

# Low Severity

## L-01 Missing Docstrings

Throughout the codebase, multiple instances of missing docstrings were identified:

- In `SolvBTCV3.sol`, the `SolvBTCV3` contract
- In `SolvBTCV3.sol`, the `DestroyBlackFunds` event
- In `SolvBTCV3.sol`, the `destroyBlackFunds` function
- In `SolvBTCV2_1.sol`, the `SolvBTCV2_1` contract
- In `SolvBTCV2_1.sol`, the `SOLVBTC_MINTER_ROLE` state variable
- In `SolvBTCV2_1.sol`, the `SOLVBTC_POOL_BURNER_ROLE` state variable

Consider thoroughly documenting all functions (and their parameters) that are part of any contract's public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

***Update:*** *Resolved in pull request #5 at commit 78e9861. The team stated:*

> *Modify according to the suggestions.*

## L-02 Missing Zero-Address Check

Within `BlacklistableUpgradeable.sol`, both the `addBlacklist` and `addBlacklistBatch` functions allow `address(0)` to be added to the `_blacklisted` mapping. If a zero address is added to the blacklist, then all mint and burn operations will fail due to the checks present in the `_update` function of the `SolvBTCV3` contract.

To avoid adding `address(0)` to the blacklist, consider validating the arguments provided to both `addBlacklist` and `addBlacklistBatch`. Alternatively, consider adding a zero-address check in `_addBlacklist`.

***Update:*** *Resolved in pull request #6 at commit 41c5be4. The team stated:*

> *Modify according to the suggestions.*

# Notes & Additional Information

## N-01 Use `calldata` Instead of `memory`

When dealing with the parameters of `external` functions, it is more gas-efficient to read their arguments directly from `calldata` instead of storing them in `memory`. `calldata` is a read-only region of memory that contains the arguments of incoming `external` function calls. This makes using `calldata` as the data location for such parameters cheaper and more efficient compared to `memory`. Thus, using `calldata` in such situations will generally save gas and improve the performance of a smart contract.

Throughout the codebase, multiple instances where function parameters should use `calldata` instead of `memory` were identified:

- In `BlacklistableUpgradeable.sol`, the `accounts_` parameter in the `addBlacklistBatch` function
- In `BlacklistableUpgradeable.sol`, the `accounts_` parameter in the `removeBlacklistBatch` function.

Consider using `calldata` as the data location for the parameters of `external` functions to optimize gas usage.

***Update:*** *Resolved in pull request #7 at commit 7d96f1e. The team stated:*

> *Modify according to the suggestions.*

## N-02 Use Custom Errors

Since Solidity version 0.8.4, custom errors provide a cleaner and more cost-efficient way to explain to users why an operation failed.

Throughout the codebase, multiple instances of `revert` and/or `require` messages were identified:

- In the `BlacklistableUpgradeable.sol` file, the `require` statement with the message "Blacklistable: caller is not the blacklist manager"
- In the `BlacklistableUpgradeable.sol` file, the `require` statement with the message "Blacklistable: account is blacklisted"
- In the `BlacklistableUpgradeable.sol` file, the `require` statement with the message "Blacklistable: invalid blacklist manager"
- In the `SolvBTCV2_1.sol` file, the `require` statement with the message "SolvBTC: mint value cannot be 0"
- In the `SolvBTCV2_1.sol` file, the `require` statement with the message "SolvBTC: burn value cannot be 0"
- In the `SolvBTCV2_1.sol` file, the `require` statement with the message "SolvBTC: burn value cannot be 0"
- In the `SolvBTCV3.sol` file, the `require` statement with the message "SolvBTCV3: account is not blacklisted".

For conciseness and gas savings, consider replacing `require` and `revert` messages with custom errors.

***Update:*** *Resolved in [pull request #21](#) at commit [829067c](#). The team stated:*

> *Modify according to the suggestions.*

## N-03 Inconsistent Order Within Contracts

Throughout the codebase, multiple instances of contracts deviating from the Solidity Style Guide due to having inconsistent function ordering were identified:

- The `BlacklistableUpgradeable` contract in `BlacklistableUpgradeable.sol`
- The `SolvBTCV2_1` contract in `SolvBTCV2_1.sol`
- The `SolvBTCV3` contract in `SolvBTCV3.sol`

To improve the project's overall legibility, consider standardizing ordering throughout the codebase as recommended by the Solidity Style Guide (Order of functions).

***Update:*** *Resolved in [pull request #22](#) at commit [e7a0e97](#), at commit [1816208](#). The team stated:*

> _

## N-04 Lack of Security Contact

Providing a specific security contact (such as an email or ENS name) within a smart contract significantly simplifies the process for individuals to communicate if they identify a vulnerability in the code. This practice is quite beneficial as it permits the code owners to dictate the communication channel for vulnerability disclosure, eliminating the risk of miscommunication or failure to report due to a lack of knowledge on how to do so. In addition, if the contract incorporates third-party libraries and a bug surfaces in those, it becomes easier for their maintainers to contact the appropriate person about the problem and provide mitigation instructions.

Throughout the codebase, multiple instances of contracts missing a security contact were identified:

- The `BlacklistableUpgradeable` abstract contract
- The `ISolvBTC` interface
- The `SolvBTCV2_1` contract
- The `SolvBTCV3` contract

Consider adding a NatSpec comment containing a security contact above each contract definition. Using the `@custom:security-contact` convention is recommended as it has been adopted by the OpenZeppelin Wizard and the ethereum-lists.

***Update:*** *Resolved in pull request #23 at commit 0cfaf24. The team stated:*

> *Modify according to the suggestions.*

## N-05 Non-Explicit Imports

The use of non-explicit imports in the codebase can decrease code clarity and may create naming conflicts between locally defined and imported variables. This is particularly relevant when multiple contracts exist within the same Solidity file or when inheritance chains are long.

Throughout the codebase, multiple instances of non-explicit imports were identified:

- The "@openzeppelin/contracts-upgradeable/access/Ownable2StepUpgradeable.sol"; import in `BlacklistableUpgradeable.sol`
- The "@openzeppelin/contracts/token/ERC20/IERC20.sol"; import in `ISolvBTC.sol`
- The "@openzeppelin/contracts/utils/introspection/IERC165.sol"; import in `ISolvBTC.sol`
- The "./external/IERC721Receiver.sol"; import in `ISolvBTC.sol`
- The "./external/IERC3525Receiver.sol"; import in `ISolvBTC.sol`
- The "@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol"; import in `SolvBTCV2_1.sol`
- The "@openzeppelin/contracts-upgradeable/utils/ReentrancyGuardUpgradeable.sol"; import in `SolvBTCV2_1.sol`
- The "@openzeppelin/contracts-upgradeable/access/Ownable2StepUpgradeable.sol"; import in `SolvBTCV2_1.sol`
- The "@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol"; import in `SolvBTCV2_1.sol`
- The "./ISolvBTC.sol"; import in `SolvBTCV2_1.sol`
- The "./SolvBTCV2_1.sol"; import in `SolvBTCV3.sol`
- The "./access/BlacklistableUpgradeable.sol"; import in `SolvBTCV3.sol`

Following the principle that clearer code is better code, consider using the named import syntax *(`import {A, B, C} from "X"`)* to explicitly declare which contracts are being imported.

***Update:*** *Resolved in pull request #12 at commit 0335695, at commit 6d8734d.*

## N-06 Gas Optimization in `BlacklistableUpgradeable.sol`

In `BlacklistableUpgradeable.sol`, multiple opportunities for optimizing loops were identified:

- **Unchecked Block for Incremental Updates**: Since Solidity `0.8.0`, arithmetic operations include overflow checks that increase gas costs. For positively incrementing variables in loops, using an unchecked block can save gas. This applies to the `i++` operation in both `addBlacklistBatch` and `removeBlacklistBatch` functions.
- **Prefix Increment Operator (`++i`) for Gas Savings**: The prefix increment operator (`++i`) is more gas-efficient than the postfix increment operator (`i++`), as it avoids storing the value before the increment. This applies to all the `i++` operations in loops.
- **Variables Initialized With Their Default Values**: In `BlacklistableUpgradeable.sol`, several variables, like `i`, are initialized with their default values. Initializing variables with default values can be wasteful in terms of gas. It is recommended to avoid unnecessarily initializing variables with their default values.

Consider using an `unchecked` block for increments or upgrading to Solidity `^0.8.22` for automatic overflow check optimizations. Switching from `i++` to `++i` can further reduce gas usage, and avoiding redundant variable initialization will prevent unnecessary gas costs.

***Update:** Resolved in pull request #25 at commit df401ab.*

## N-07 Function Visibility Overly Permissive

Throughout the codebase, multiple instances of functions having overly permissive visibility were identified:

- The `_getBlacklistableStorage` function in `BlacklistableUpgradeable.sol` with `internal` visibility could be limited to `private`.
- The `_addBlacklist` function in `BlacklistableUpgradeable.sol` with `internal` visibility could be limited to `private`.
- The `_removeBlacklist` function in `BlacklistableUpgradeable.sol` with `internal` visibility could be limited to `private`.

To better convey the intended use of functions and to potentially realize some additional gas savings, consider changing a function's visibility to be only as permissive as required.

***Update:*** *Resolved in pull request #14 at commit 962b317, at commit 166734f.*

## N-08 Typographical Error

Typographical errors can negatively affect code clarity. This comment in line 19 of the `SolvBTCV2_1.sol` contract should say "variables" instead of "virables".

Consider correcting the above typographical error to improve the readability of the codebase.

***Update:*** *Resolved in pull request #15 at commit e3e3340. The team stated:*

> *Modify according to the suggestions.*

## N-09 Unclear Comment Block

The block of commented-out code spanning lines 111-145 of `SolvBTCV2.sol` has no explanation to match similar blocks above. Adding an explanation for this commented-out code will help reduce confusion in the future and prevent developer intentions from being forgotten.

Consider adding an explanation above the commented-out code block like the one found in lines 18-21 or 49-52.

***Update:*** *Resolved in pull request #17 at commit 001dc64. The team stated:*

> *Modify according to the suggestions.*

## N-10 Hardcoded Addresses

In `SolvBTCV2_1.sol`, multiple instances of hardcoded address values were identified:

- The `0x55C09707Fd7aFD670e82A62FaeE312903940013E` value
- The `0x55C09707Fd7aFD670e82A62FaeE312903940013E` value

Consider declaring hardcoded addresses as `immutable` and assigning them via constructor arguments. This allows the code to remain the same across deployments on different networks and avoids error-prone address hardcoding and recompilation when addresses need to change.

***Update:*** *Resolved in pull request #18 at commit faf9b5b. The team stated:*

> *We have made the changes as per the suggestions.*

## N-11 Incorrect Interface Implementation

The `SolvBTCV2_1` contract does not strictly implement the `ISolvBTC` interface as required. The parameter names in the implementation differ from those in the interface:

- The `mint` function uses `address account_` and `uint256 value_`, whereas the interface specifies `address account` and `uint256 value`.
- The `burn` function uses `address account_` and `uint256 value_`, whereas the interface specifies `address account` and `uint256 value`.
- The `burn` function uses `uint256 value_`, whereas the interface specifies `uint256 value`.

Although the functionality remains unaffected, consider aligning parameter names with the interface to improve clarity and reduce potential confusion.

***Update:*** *Resolved in pull request #19 at commit 1e94c85. The team stated:*

> *Modify according to the suggestions.*

## N-12 Unnecessary Interface ID Reference

In the `SolvBTCV2_1` contract, the `supportsInterface` function explicitly checks for `type(IERC165).interfaceId`, even though `AccessControlUpgradeable` already provides support for `IERC165`.

Consider removing the redundant `type(IERC165).interfaceId` reference to simplify the code and reduce confusion.

***Update:*** *Resolved in pull request #20 at commit f47c81f. The team stated:*

> *Modify according to the suggestions.*

## N-13 Specify Versions for Imports

The various OpenZeppelin libraries imported in the codebase do not specify a version number. This means that these dependencies may change over time.

The un-versioned OpenZeppelin imports can be found in the following files:

- `BlacklistableUpgradeable.sol`
- `SolvBTCV2_1.sol`
- `ISolvBTC.sol`

Consider specifying versions for all external dependencies. This will ensure that all deployments of the codebase are identical and that no functionality is lost in the event a breaking change is introduced in a future version of these libraries.

***Update:*** *Resolved in pull request #24 at commit 6cc5c0e. The team stated:*

> *Modify according to the suggestions.*

## Conclusion

This audit assessed the implementation of the SolvBTC contracts, focusing on their ERC-20 upgradeable design, blacklist functionality, and role-based access controls. Each finding was accompanied by actionable recommendations aimed at further enhancing the security, consistency, and clarity of the codebase.

The Solv team was highly collaborative and responsive throughout the audit process, demonstrating a strong commitment to improving the project. The codebase already reflects a solid adherence to best practices, and by addressing the identified issues, its overall robustness and maintainability will be further strengthened.