OpenZeppelin | security

Solv Mainnet Minter Audit

November 27, 2024

Table of Contents

Summary

Type	Bridge
Timeline	From 2024-11-25 To 2024-11-26
Languages	Solidity

Total Issues	7 (0 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	3 (0 resolved)
Notes & Additional Information	4 (0 resolved)
Client Reported Issues	0 (0 resolved)

Scope

We audited the solv-finance/mainnet-minter-contracts repository at the `cbf8a53` commit.

In scope was the following file:

```
contracts
└── MainnetMinter.sol
```

System Overview

Solv protocol provides a bridge between Bitcoin and EVM-based blockchains, allowing users to mint `solvBTC` tokens and derivatives on any supported chain by depositing Bitcoin on the Bitcoin mainnet. This process begins with users linking their EVM and Bitcoin wallets to associate their EVM address with a Bitcoin address. Users then initiate a Bitcoin transaction to the protocol's vault, embedding necessary metadata such as their EVM chain ID, wallet address and token to mint in the Bitcoin transaction's `OP_RETURN` field. This metadata ensures that the deposited Bitcoin can be securely associated with the user's wallet for subsequent minting on any of the supported chains.

The backend monitors and indexes Bitcoin transactions to detect deposits in the protocol vault. Once a transaction is verified and deemed compliant with risk control rules, it generates an approval signature. This signature, combined with the Bitcoin transaction details, allows users to invoke the Mainnet Minter contract on the supported chains. The Mainnet Minter contract verifies the data and signature using EIP-712 and ensures that the Bitcoin transaction hash is unique and valid. If all checks pass, the contract mints the corresponding amount of tokens to the user's provided address. This system ensures a secure, decentralized, and transparent mechanism for bridging Bitcoin assets into the EVM-based blockchains.

Trust Assumptions and Privileged Roles

- The Mainnet Minter contract will be deployed on multiple EVM chains, including Ethereum, BNB Chain, Avalanche, and more with the assumption that these chains operate reliably and securely without significant disruptions or vulnerabilities.
- Solv's governance multisig, a 3-out-of-5 wallet, owns the Mainnet Minter contract and has the authority to manage privileged parameters, such as updating the signer, modifying the allowed token list, and enabling or disabling push functionality.
- The off-chain component responsible for verifying Bitcoin transactions is trusted to function securely and correctly, ensuring that only valid Bitcoin transactions are approved and signatures are not generated for invalid or malicious transactions.

Low Severity

L-01 Limited Authorization Flexibility Due to Single Signer Support

The `MainnetMinter` contract is designed to utilize a single signer for transaction verification, as indicated by the contract's code. This approach restricts the contract's ability to accommodate various authorization levels, which could be beneficial in scenarios that demand differentiated permissions. For instance, employing externally owned accounts (EOAs) for transactions involving smaller amounts and utilizing multisignature wallets for those entailing larger sums could significantly enhance security and operational flexibility. The current design's lack of support for multiple signers with distinct roles or caps limits the contract's adaptability and complicates the implementation of nuanced control mechanisms based on transaction characteristics such as value or associated risk.

To address this limitation and bolster the contract's security and versatility, it is advisable to integrate functionality that allows for the specification of multiple signers, each associated with unique caps or roles. This could be achieved through the introduction of a mapping structure that links signers to their respective authorization parameters, thereby facilitating dynamic transaction verification that aligns with the nature and requirements of each transaction. Such an enhancement would not only improve the contract's security posture by enabling more granular control over transaction approvals but also increase its operational flexibility without necessitating multiple instances of the `MainnetMinter` contract.

L-02 Risk of Failed Deployments with Solidity Version `0.8.27`

The `MainnetMinter` contract uses Solidity version `0.8.27`. This version utilises two opcodes which are not supported by all EVM chains and L2s: `PUSH0` and `MCOPY`. For example, `MCOPY` is currently not supported on Avalanch C-Chain and Linea. Deployment of the `MainnetMinter` contract on such chains will result in the transaction being reverted.

Since this contract will be deployed on multiple chains, consider using an older EVM version while compiling for chains that do not support the new opcodes. A comparison of supported opcodes by chain can be found on evmdiff.

L-03 Use of `call` instead of `staticcall` in EIP-1271 Signature Verification

EIP-1271 allows smart contract wallets to issue signatures using the `isValidSignature(bytes32,bytes)` interface. The `isValidSignature` is prescribed to be a `view` function in the EIP and is supposed to be called using a `staticcall`. A `staticcall` reverts if a state change is made during the life of the call. The `MainnetMinter` contract uses the `call` opcode to verify the smart wallet signature.

Consider using the `isValidSignatureNow` function of the `SignatureChecker` contract of the OpenZeppelin Contracts library which verifies signatures for both EOAs and smart contract wallets.

Notes & Additional Information

N-01 Implementation Contract Initializer Not Disabled

An implementation contract in a proxy pattern allows anyone to call its `initialize` function. While not a direct security concern, preventing the implementation contract from being initialized is important, as this could allow an attacker to take over the contract. This would not affect the proxy contract's functionality, as only the implementation contract's storage would be affected.

In `MainnetMinter.sol`, in the initializable contract `MainnetMinter`, `_disableInitializers()` is not called in the constructor.

Consider calling `_disableInitializers()` in initializable contract constructors to prevent malicious actors from front-running initialization.

N-02 Multiple Optimizable State Reads

In the file `MainnetMinter.sol` there are multiple optimizable storage reads:

- The `signer` storage read on lines 86, 89 and 92.

Consider reducing `SLOAD` operations that consume unnecessary amounts of gas by caching the value in a stack variable.

N-03 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.

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.

N-04 Absence of EIP-7201 Compliance for Safer Storage Management

The current contract architecture overlooks the implementation of EIP-7201: Namespaced Storage Layout, a critical standard for structuring storage in a manner that safeguards against collisions in upgradeable contracts. This standard is instrumental in ensuring that storage variables are meticulously organized within distinct namespaces, thereby significantly mitigating the risk of storage collisions. Such collisions can occur during contract upgrades or through interactions with inherited contracts, potentially resulting in erratic contract behavior or the introduction of exploitable vulnerabilities.

The absence of EIP-7201 compliance exposes the contract to unnecessary risks associated with future modifications or enhancements. As contracts evolve and storage structures become more complex, the likelihood of inadvertently overwriting storage slots increases, which could compromise the integrity and security of the contract.

To fortify the contract's upgradeability framework and enhance its resilience against such risks, adopting a namespaced storage layout as outlined in EIP-7201 is strongly recommended. By doing so, storage variables will be encapsulated within unique namespaces, effectively isolating them and preventing any unintended interactions between disparate storage

elements. This approach not only streamlines the process of upgrading contracts by clearly delineating storage boundaries but also significantly improves the contract's maintainability and security posture. Implementing a namespaced storage layout will serve as a proactive measure to safeguard against potential storage-related issues, ensuring a more robust and reliable contract infrastructure.

Conclusion

`SolvBTC` is a tokenized representation of Bitcoin designed to bring Bitcoin liquidity into Ethereum and other EVM chains. The `MainnetMinter` contract will be used to verify and mint `SolvBTC` against bitcoin. The contract was well-written and thoroughly documented and no major issues were found. The Solv Protocol team was very responsive and provided us with extensive documentation about the project.