Blockchain
Security Audit Report

Prepared for BEVM Foundation

Prepared by Supremacy

March 21, 2024

Contents

1 Introduction

Given the opportunity to review the design document and related codebase of the BEVM, we outline in the report our systematic approach to evaluate potential security issues in the blockchain implementation, and provide additional suggestions or recommendations for improvement. Our results show that the given version of blockchain can be further improved due to the presence of several issues related to either security or performance. This document outlines our audit results.

1.1 About Client

BEVM is the first EVM-compatible Bitcoin layer2 based on Taproot Consensus. It allows all DApps which can run in the Ethereum ecosystem to operate on Bitcoin L2.

Item	Description
Client	BEVM Foundation
Website	https://bevm.io
Type	Blockchain
Languages	Rust

1.2 Audit Scope

In the following, we show the Git repository of reviewed file and the commit hash used in this security audit:

- Repository: https://github.com/btclayer2/BEVM-DEV
- Commit Hash: 4edb88e1cc09e792203451b9b20e9e77858e14bf

And this is the commit hash after all fixes for the issues found in the security audit have been checked in:

- Repository: https://github.com/btclayer2/BEVM-DEV
- Commit Hash: 36301d7e43170a80632aff6f5d9bc3ae65e8b527

1.3 Changelogs

Version	Date	Description
0.1	March 18, 2024	Initial Draft
1.0	March 21, 2024	Final Release

1.4 About Us

Supremacy is a leading blockchain security firm, composed of industry hackers and academic researchers, provide top-notch security solutions through our technology precipitation and innovative research.

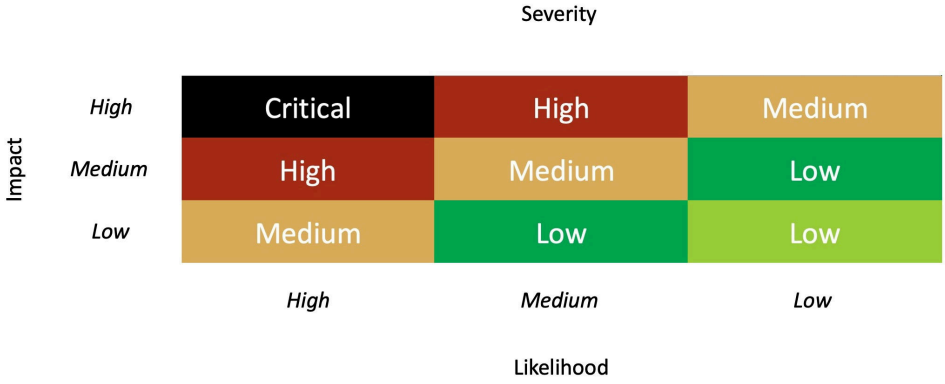
We are reachable at Twitter (https://twitter.com/SupremacyHQ), or Email (contact@supremacy.email).

1.5 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- Likelihood represents the likelihood of a finding to be triggered or exploited in practice
- Impact specifies the technical and business-related consequences of a finding
- Severity is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.



As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

2 Findings

The table below summarizes the findings of the audit, including status and severity details.

ID	Severity	Description	Status
1	Medium	The potential numeric bounds saturation	Fixed
2	Medium	Lack of transactional macro	Fixed
3	Informational	Centralized risk	Confirmed

2.1 Medium

1. The potential numeric bounds saturation [Medium]

`Severity: Medium` `Likelihood: Medium` `Impact: Medium`

`Status: Fixed`

Description:

In the codebase, there are a large number of saturating mathematical computing functions. In some special cases, it may lead to the expected results.

Include `saturating_mul`, `saturating_add`, and `saturating_sub` functions.

Because of the own characteristics of these functions, it may cause saturating at the numeric bounds instead of overflowing, the returned result is inaccurate.

```
138    fn set_withdrawal_state_list(u: u32, ) -> Weight {
139      // Proof Size summary in bytes:
140      //  Measured:  `429`
141      //  Estimated: `3894`
142      // Minimum execution time: 54_037_000 picoseconds.
143      Weight::from_parts(55_489_498, 3894)
144        // Standard Error: 789
145        .saturating_add(Weight::from_parts(906, 0).saturating_mul(u.into()))
146        .saturating_add(T::DbWeight::get().reads(8_u64))
147        .saturating_add(T::DbWeight::get().writes(7_u64))
148    }
```

weights.rs

Recommendation: Revise the code, use more secure functions such as `checked_mul`, `checked_add`, `checked_sub`.

2. Lack of transactional macro [Medium]

`Severity: Medium` `Likelihood: Medium` `Impact: Medium`

`Status: Fixed`

Description:

`#[transactional]` must be used for every extrinsic in the xpallet, otherwise the state will not be canceled on revert.

In the following files, there are some functions that lack Transactional macro modified function.

```
./xpallets/assets-bridge/src/lib.rs
./xpallets/gateway/bitcoin/src/lib.rs
./xpallets/gateway/common/src/lib.rs
./xpallets/gateway/records/src/lib.rs
```

Without transactional after revert the balance record will not be deleted.

```rust
411     /// Allow root or trustees could remove pending deposits for an address and
    decide whether
412     /// deposit to an account id. if pass `None` to `who`, would just remove
    pending, if pass
413     /// Some, would deposit to this account id.
414     #[pallet::call_index(10)]
415     #[pallet::weight(< T as Config >::WeightInfo::remove_pending())]
416     pub fn remove_pending(
417       origin: OriginFor<T>,
418       addr: BtcAddress,
419       who: Option<OpReturnAccount<T::AccountId>>,
420     ) -> DispatchResult {
421       <T as Config>::CouncilOrigin::try_origin(origin)
422         .map(|_| ())
423         .or_else(ensure_root)?;
424
425       if let Some(w) = who {
426         remove_pending_deposit::<T>(&addr, &w);
427       } else {
428         log!(info, "[remove_pending] Release pending deposit directly, not
    deposit to someone, addr:{:?}", try_addr(&addr));
429         PendingDeposits::<T>::remove(&addr);
430       }
431       Ok(())
432     }
```

lib.rs

Recommendation: Revise the `#[transactional]` macro to the necessary functions.

2.2 Informational

3. Centralized risk [Informational]

`Status: Confirmed`

Description:

In the codebase, there is a privilege account, which has the right to perform privileged operations directly after passing ensure_root() access control.

Our analysis shows that privileged accounts need to be scrutinized. In the following, we will examine privileged accounts and the associated privileged access in the current blockchain.

Note that if the privileged owner account is a plain wallet account, this may be worrisome and pose counter-party risk to the blockchain users. A multi-sig account could greatly alleviate this concern, though it is still far from perfect. Specifically, a better approach is to eliminate the administration key concern by transferring the role

to a community-governed DAO. In the meantime, a timelock-based mechanism can also be considered as mitigation.

```
293       #[pallet::weight({0})]
294       #[pallet::call_index(5)]
295       #[transactional]
296       pub fn direct_deposit_btc(
297         origin: OriginFor<T>,
298         txid: H256,
299         evm_account: H160,
300         sats_amount: u64,
301       ) -> DispatchResultWithPostInfo {
302         ensure_root(origin)?;
303
304         Self::apply_direct_deposit_btc(txid.to_fixed_bytes(), evm_account,
      sats_amount)?;
305
306         Ok(Pays::No.into())
307       }
308
309       #[pallet::weight({0})]
310       #[pallet::call_index(6)]
311       #[transactional]
312       pub fn call_contract(
313         origin: OriginFor<T>,
314         contract: H160,
315         inputs: Vec<u8>,
316       ) -> DispatchResultWithPostInfo {
317         ensure_root(origin)?;
318
319         Self::call_evm(contract, inputs)?;
320
321         Ok(Pays::No.into())
322       }
```

lib.sol

Recommendation: Initially onboarding could can use multisign wallets or timelocks to initially mitigate centralization risks, but as a long-running protocol, we recommend eventually transfer the privileged account to the intended DAO-like governance. All changed to privileged operations may need to be mediated with necessary timelocks.

Eventually, activate the normal on-chain community-based governance life-cycle and ensure the intended trustless nature and high-quality distributed governance.

3 Disclaimer

This security audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. This security audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the blockchain, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues, also cannot make guarantees about any additional code added to the assessed project after the audit version. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of blockchain(s). Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.