SUPREMACY

Smart Contract Security Audit Report

Prepared for Satoshi Protocol

Prepared by Supremacy

March 19, 2024

Contents

1 Introduction

Given the opportunity to review the design document and related codebase of the Satoshi Protocol, we outline in the report our systematic approach to evaluate potential security issues in the smart contract(s) implementation, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts 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

The Satoshi Protocol aims to provide a cornerstone for DeFi and make BTC truly spendable in daily usage by offering a CDP-style stablecoin. The Satoshi Protocol introduces SAT, a BTC-collateralized stablecoin, using the BEVM and a CDP model to enhance Bitcoin's use in DeFi while maintaining its decentralized ethos.

Item	Description
Client	Satoshi Protocol
Website	https://satoshiprotocol.org
Type	Smart Contract
Languages	Solidity
Platform	EVM-compatible

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/Satoshi-Protocol/satoshi-core/tree/main/src
- Commit Hash: 7973071d32e905e648e5efda8f93bbc68bc45765

Below are the files in scope for this security audit and their corresponding MD5 hashes.

Filename	MD5
./OSHI/CommunityIssuance.sol	7c20ce0ed0710683c3f6b5c9a82aa576
./OSHI/OSHIToken.sol	815c09fd3a7f91e9439ae996a3b67b1f
./OSHI/RewardManager.sol	c8bb93a9e1f6b7f82da6f2287a4f6c32
./core/BorrowerOperations.sol	40443e38773b3014894730d16940303f
./core/DebtToken.sol	1713545cd59c91167b562e86d1f52a28
./core/Factory.sol	8b3eabfa986f866c2814ceedc1714fa0
./core/GasPool.sol	c560234264e1005a75c534d1834e62fc
./core/LiquidationManager.sol	5139940f107c6aff2fc83d6d3cc342b7
./core/PriceFeedAggregator.sol	c5f08f5917de84c9b0cf1b370c05d85f
./core/SatoshiCore.sol	7987a032937d37bd6359a507cb2078ca
./core/SortedTroves.sol	4defcd42abb672848ca38cd1e40b869e
./core/StabilityPool.sol	f8426606e43e8207bffdf1079d4aa7b4
./core/TroveManager.sol	9b50e2e3b06ea106fc3fe71f0b1253cb
./dependencies/DelegatedOps.sol	1ea9975011cb43bae533bc5bad51f601
./dependencies/SatoshiBase.sol	f52b316690bf695ab7317f899d777dc6
./dependencies/SatoshiMath.sol	0a7e0cca269df12964715586e831b99f
./dependencies/SatoshiOwnable.sol	87970ace84ae929f8ada8cecfd1c1e65
./dependencies/priceFeed/PriceFeedChainlink.sol	2957c82cdeee5444cf367b57440d2273
./dependencies/priceFeed/PriceFeedDIAOracle.sol	321b201213f52bfbad2684208e884dc3

./helpers/MultiCollateralHintHelpers.sol	2b9d0cc3825e4dcfd842fe63ad3326cc
./helpers/MultiTroveGetter.sol	52a3a0163a9129e3cd768abf52e5a3e6
./helpers/SatoshiBORouter.sol	7203311aa26a0273c7a5f9a93038b983
./helpers/TroveManagerGetters.sol	fedaaa5f974e013075e6ba71d0b2ca56

1.3 Changelogs

Version	Date	Description
0.1	March 18, 2024	Initial Draft
1.0	March 19, 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.

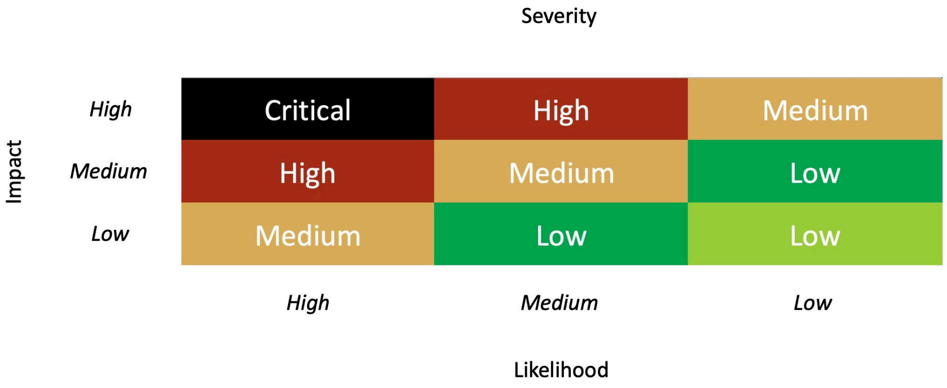
We are reachable at Twitter (https://twitter.com/SupremacyHQ), or Email (contact@supremacy.email).

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.



Severity

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

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 permit function lack address validation	Fixed
2	Medium	Centralized risk	Fixed
3	Low	The potential fake token setters	Fixed

2.1 Medium

1. The permit function lack address validation [Medium]

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

`Status: Fixed`

Description:

Validation that `recoveredAddress` is not a zero address is missing in the `OSHIToken::permit()` function. If the zero address is allowed to hold the OSHIToken, an attacker can get those assets.

```
74      function permit(address owner, address spender, uint256 amount, uint256 deadline, uint8 v, bytes32 r, bytes32 s)
75          external
76          override
77      {
78          require(deadline >= block.timestamp, "OSHI: expired deadline");
79          bytes32 digest = keccak256(
80              abi.encodePacked(
81                  "\x19\x01",
82                  domainSeparator(),
83                  keccak256(abi.encode(permitTypeHash, owner, spender, amount, _nonces[owner]++, deadline))
84              )
85          );
86          address recoveredAddress = ecrecover(digest, v, r, s);
87          require(recoveredAddress == owner, "OSHI: invalid signature");
88          _approve(owner, spender, amount);
89      }
```

OSHIToken.sol

Recommendation: Revise the `#L87` code logic as `require(recoveredAddress == owner && recoveredAddress != address(0), "OSHI: invalid signature");");`

2. Centralized risk [Medium]

`Severity: Medium` `Likelihood: Low` `Impact: High`

`Status: Fixed`

Description:

In the Satoshi protocol, there is a privilege account, which has the right to directly transfer a specific asset in the reward manager.

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 contract.

Note that if the privileged owner account is a plain EOA, this may be worrisome and pose counter-party risk to the protocol 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.

```
    function transferToken(IERC20 token, address receiver, uint256 amount) external onlyOwner {
        token.safeTransfer(receiver, amount);
    }

    function setTokenApproval(IERC20 token, address spender, uint256 amount) external onlyOwner {
        token.safeApprove(spender, amount);
    }
```

RewardManager.sol

Recommendation: Revise `transferToken()` & `setTokenApproval()` code logic as token verification.

2.2 Low

3. The potential fake token setters [Low]

Severity: Low Likelihood: Low Impact: Low

Status: Fixed

Description:

In the `CommunityIssuance::setAddresses()` function, state variables such as OSHIToken & stabilityPool are set, and its main function is to claim allocations set by ownership privileged accounts. however, the setAddresses() function does not have any limitations, which means that it can be set multiple times, both before and after the claim. This means that if the owner is a malicious actor, then the claim is not an OSHIToken.

```
    function setAddresses(IOSHIToken _oshiToken, IStabilityPool _stabilityPool) external onlyOwner {
        OSHIToken = _oshiToken;
        stabilityPool = _stabilityPool;

        emit OSHITokenSet(_oshiToken);
        emit StabilityPoolSet(_stabilityPool);
    }
```

CommunityIssuance.sol

Recommendation: Revise `setAddresses()` code logic as called only once.

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 smart contracts, 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 smart contract(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.