Satoshi Protocol
Audit Report

Fri Mar 15 2024

contact@scalebit.xyz https://twitter.com/scalebit_

ScaleBit

Satoshi Protocol Audit Report

1 Executive Summary

1.1 Project Information

Description	The Satoshi Protocol aims to provide a cornerstone for DeFi and make BTC truly spendable in daily usage by offering a CDP-style stablecoin.
Type	DeFi
Auditors	ScaleBit
Timeline	Sun Feb 25 2024 - Fri Mar 15 2024
Languages	Solidity
Platform	BEVM
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/Satoshi-Protocol/satoshi-core
Commits	17b598ea88f0057078dfb0e477785842ca1d6a2b

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
SPO	src/core/StabilityPool.sol	bf8a784b8c6d27241dd6d91e3c479bbc437bdf5e
BOP	src/core/BorrowerOperations.sol	3319ddaea56923902a70da998c4b4995b92e86f4
DTO	src/core/DebtToken.sol	1fcb3624ebed73ca64c2fa84af27be6579d4973a
PFA	src/core/PriceFeedAggregator.sol	35dc2309674076c659bcebbdd5e9c0806889aded
SCO	src/core/SatoshiCore.sol	7ebc6fcf7e11dff04a653161c7d7f3d4d737e4a1
GPO	src/core/GasPool.sol	6ed2172b8bd4a679ac642f31ebf182a27170a167
STR	src/core/SortedTroves.sol	214ecf358aea7d10cac55d1406a8d609c32316d4
FAC	src/core/Factory.sol	c2ddda738b92998aabbed70b3a44a75112c16774
TMA	src/core/TroveManager.sol	6dfbe13184eb91bf8ef35d3818010370fa37453d
LMA	src/core/LiquidationManager.sol	1eedbb92bb8507da6222e4a397e4cf283de58ade
SMA	src/dependencies/SatoshiMath.sol	fbc2e4abf44176b403937360cedc12060096f00e

SOW	src/dependencies/SatoshiOwnable.sol	e0317952f3df16993eee98b52b93cf3cb370df8a
DOP	src/dependencies/DelegatedOps.sol	f364d8839f3f048406b39f36f20aad2abbe455de
SBA	src/dependencies/SatoshiBase.sol	bb39e16b217c0c0ac5b6bb87c92943373b6e53f6
PFDIAO	src/dependencies/priceFeed/PriceFeedDIAOracle.sol	e61989dbcb20f0d303f33ef76847023efeec7f9f
PFC	src/dependencies/priceFeed/PriceFeedChainlink.sol	6dacb4b2c6f3f22400cf80440fd2af701f006028
MTG	src/helpers/MultiTroveGetter.sol	491e8eeb416d85b2b20ca10973b41124d20ef46d
SBOR	src/helpers/SatoshiBORouter.sol	1016de564896425213b75a9a23d0723d0b730c1c
MCHH	src/helpers/MultiCollateralHintHelpers.sol	db73e853c2510d32af53c55f3ddb683fdc8d9622
TMG	src/helpers/TroveManagerGetters.sol	96b9b26e005092d6f662093e426be6037d29cec8

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	6	6	0
Informational	2	2	0
Minor	2	2	0
Medium	1	1	0
Major	1	1	0
Critical	0	0	0

1.4 ScaleBit Audit Breakdown

ScaleBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow
- Number of rounding errors
- Unchecked External Call
- Unchecked CALL Return Values
- Functionality Checks
- Reentrancy
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic issues
- Gas usage
- Fallback function usage
- tx.origin authentication
- Replay attacks
- Coding style issues

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by Satoshi Protocol to identify any potential issues and vulnerabilities in the source code of the Satoshi Protocol smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 6 issues of varying severity, listed below.

ID	Title	Severity	Status
BOP-1	`require()` / `revert()` Statements Should Have Descriptive Reason Strings	Minor	Fixed
BOP-2	Use `!= 0` instead of `> 0` for Unsigned Integer Comparison	Informational	Fixed
FAC-1	Use `Calldata` Instead of `Memory` for Function Arguments That Do not Get Mutated	Informational	Fixed
PFA-1	`Initialize` Could Be Front-Run	Major	Fixed
SBO-1	Use `SafeTransfer/SafeTransferFrom` Consistently Instead of `Transfer/TransferFrom`	Medium	Fixed
STR-1	Missing Events for Key Operations	Minor	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the Satoshi Protocol Smart Contract :

Admin

- `admin` can create a new instance through `deployNewInstance`.
- `admin` can set the reward rate through `setRewardRate`.
- `admin` can set the minNetDebt through `setMinNetDebt`.
- `admin` can set the collateral through `configureCollateral`.
- `admin` can set the troveManager useable through `enableTroveManager`.
- `admin` can set the priceFeed through `setPriceFeed`.
- `admin` can set the fee receiver through `setFeeReceiver`.
- `admin` can set the guardian through `setGuardian`.
- `admin` can set the reward manager through `setRewardManager`.
- `admin` can change the paused parameter through `setPaused`.
- `admin` can transfer ownership through `commitTransferOwnership`.
- `admin` can revoke transfer ownership through `revokeTransferOwnership`.
- `admin` can start the sunset through `startSunset`.
- `admin` can change the rewardRate through `setRewardRate`.
- `admin` can start sunsetting collateral through `startCollateralSunset`.
- `admin` can set the time when the OSHI claim starts through `setClaimStartTime`.
- `admin` can change the maxTimeThreshold through `updateMaxTimeThreshold`.

User

- `user` can go to flash loan through `flashLoan`.
- `user` can approve others to use tokens through `approve`.
- `user` can send collateral to a trove through `addColl`.
- `user` can withdraw collateral through `withdrawColl`.

- `user` can withdraw debt tokens from a trove through `withdrawDebt`.
- `user` can repay Debt tokens to a Trove through `repayDebt`.

4 Findings

BOP-1 `require()` / `revert()` Statements Should Have Descriptive Reason Strings

Severity: Minor

Status: Fixed

Code Location:

src/core/BorrowerOperations.sol#98;

src/core/DebtToken.sol#93,101;

src/core/TroveManager.sol#221-226;

src/mocks/WETH9.sol#32,53,56

Descriptions:

`require()` / `revert()` Statements Should Have Descriptive Reason Strings

```
require(_minNetDebt > 0);
```

```
require(msg.sender == address(borrowerOperations));
```

and so on.

Suggestion:

It is recommended to add reason strings to `require()` or `revert()`.

Resolution:

This issue has been fixed. The client has already added descriptive reason strings to `require` statements.

BOP-2 Use `!= 0` instead of `> 0` for Unsigned Integer Comparison

Severity: Informational

Status: Fixed

Code Location:

src/core/BorrowerOperations.sol#238,255,271,305,336,446

Descriptions:

When dealing with unsigned integer types, comparisons with `!= 0` are cheaper than with `> 0`.

```
if (troveManager.interestPayable() > 0) {
  troveManager.collectInterests();
}
```

Suggestion:

It is recommended to use `!= 0` instead of `> 0` for unsigned integer comparison.

Resolution:

This issue has been fixed.

FAC-1 Use `Calldata` Instead of `Memory` for Function Arguments That Do not Get Mutated

Severity: Informational

Status: Fixed

Code Location:

src/core/Factory.sol#70

Descriptions:

Mark data types as `calldata` instead of memory where possible. This makes it so that the data is not automatically loaded into memory. If the data passed into the function does not need to be changed (like updating values in an array), it can be passed in as `calldata`. The one exception to this is if the argument must later be passed into another function that takes an argument that specifies memory storage.

```
function deployNewInstance(IERC20 collateralToken, IPriceFeed priceFeed,
DeploymentParams memory params)
    external
    onlyOwner
```

Suggestion:

It is recommended to use `calldata` instead of `memory`.

Resolution:

This issue has been fixed. The client has already used `calldata` instead of `memory`.

PFA-1 `Initialize` Could Be Front-Run

Severity: Major

Status: Fixed

Code Location:

src/core/PriceFeedAggregator.sol#36-39;

src/core/BorrowerOperations.sol#78-91;

src/core/LiquidationManager.sol#48-61;

src/core/StabilityPool.sol#102-114

Descriptions:

In the contract, by calling the `initialize` function to initialize the contracts, there is a potential issue that malicious attackers preemptively call the initialize function to initialize and there is no access control verification for the initialize functions.

Suggestion:

It is suggested that the `initialize` function can be called only by privileged addresses or be called in the same transaction immediately after the contract is created to avoid being maliciously called by the attacker.

SBO-1 Use `SafeTransfer/SafeTransferFrom` Consistently Instead of `Transfer/TransferFrom`

Severity: Medium

Status: Fixed

Code Location:

src/helpers/SatoshiBORouter.sol#194,208,215,221

Descriptions:

Some tokens do not implement the ERC20 standard properly but are still accepted by most code that accepts ERC20 tokens. For example Tether (USDT)'s `transfer()` and `transferFrom()` functions on L1 do not return booleans as the specification requires, and instead have no return value. When these sorts of tokens are cast to `IERC20`, their function signatures do not match and therefore the calls made, revert (see this link for a test case).

Suggestion:

I am not certain about the token type used here. Even though the current token does not pose any issues, to account for scalability and potential uncertainties in the future, we recommend using `SafeTransfer` / `SafeTransferFrom` consistently instead of `transfer` / `transferFrom`.

Resolution:

This issue has been fixed. The client has already use `SafeTransfer/SafeTransferFrom` instead of `Transfer/TransferFrom`.

STR-1 Missing Events for Key Operations

Severity: Minor

Status: Fixed

Code Location:

src/core/SortedTroves.sol#30-33;

src/core/StabilityPool.sol#173-178;

src/core/TroveManager.sol#155-247

Descriptions:

The contract performs several key state-changing operations, such as `setConfig`, `startCollateralSunset`, `setPaused`, and so on. Although some events are present, there is a noticeable absence of event triggers in other significant functions. This absence includes operations like updates to the contract's operational status. The lack of event logs makes it difficult for external observers to track the contract's activities and state changes, reducing the contract's transparency and traceability.

Suggestion:

It's recommended to add event triggers for all key operations, including financial transactions and state changes, within the smart contract.

Resolution:

This issue has been fixed. The client has already added events for the key operations.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.