verichains

SECURITY AUDIT OF

SATOSHI PREPS

SATOSHI

Public Report

Oct 29, 2024

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Sei	Sei is an open-source, layer-1 blockchain offering infrastructure for trading apps of different types, including non-fungible token marketplaces, gaming DEXs, and DeFi DEXs. The chain conducts market-based parallelization and has a native order-matching engine that allows exchange teams to use Sei's twin-turbo consensus.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Oct 29, 2024. We would like to thank the Satoshi Preps for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the Satoshi Preps. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified no vulnerable issue in the smart contract.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Satoshi Preps

Satoshi Preps is a protocol designed to bring Bitcoin-backed perpetual contracts to the blockchain. Its mission is to enhance capital efficiency for Bitcoin holders while maintaining their exposure to BTC. Satoshi Preps achieves this by:

- Enabling Bitcoin holders to earn yields on their BTC without losing directional exposure.
- Facilitating on-chain perpetual trading using Bitcoin-based assets as collateral.
- Providing simplified and scalable liquidity through Bitcoin-denominated liquidity pools.

1.2. Audit Scope

The Satoshi Preps is a fork derived from the following sources:

- The `GMX` protocol, specifically commit *58c4ac801a95236db24c0857f820bdf4935d5814* from the *https://github.com/gmx-io/gmx-contracts*.
- The `VenusProtocolOracle`, specifically commit *7e8cbb45de85728813dbe8151257fb22a692e10b* from the *https://github.com/VenusProtocol/oracle*.

The security audit was conducted on commit `ea6439f8445dc4e8f48ae3ed006ad31417524cd9` from the *https://github.com/NLX-Protocol/NLX-1.5-contracts* repository.

1.3. Audit Methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that were considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert

- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Satoshi Preps acknowledges that the security services provided by Verichains, are conducted to the best of their professional abilities but cannot guarantee 100% coverage of all security vulnerabilities. Satoshi Preps understands and accepts that despite rigorous auditing, certain vulnerabilities may remain undetected. Therefore, Satoshi Preps agrees that Verichains shall not be held responsible or liable, and shall not be charged for any hacking incidents that occur due to security vulnerabilities not identified during the audit process.

1.5. Acceptance Minute

This final report served by Verichains to the Satoshi Preps will be considered an Acceptance Minute. Within 7 days, if no any further responses or reports is received from the Satoshi Preps, the final report will be considered fully accepted by the Satoshi Preps without the signature.

2. AUDIT RESULT

2.1. Overview

The Satoshi Preps is a modified fork of the `GMX` protocol, inheriting core features and incorporating several key updates:

2.1.1. Vault

- **Increased Leverage**: Max leverage raised from `50x` to `200x`.
- **Flexible Collateral Requirements**: Collateral tokens are no longer restricted to stable tokens for opening short positions; users can now use any token as collateral.

2.1.2. VaultPriceFeed

- **Updated Oracle Source**: Replaces GMX's default oracle with a custom price feed from the Oracle implementation in the Venus Protocol.

2.1.3. Peripheral Contracts

- **GmxTimeLock**: Modifies oracle interaction by using a custom resilient oracle from Venus Protocol rather than the default GMX oracle.
- **VaultReader**: Provides users with public access to maximum global size data via a new function, improving upon partial data availability in the `getVaultTokenInfoV4` function.

2.1.4. Oracle Modifications

The Satoshi Preps now leverages a custom resilient oracle from Venus Protocol, offering enhanced security and reliability over GMX's default oracle. Details of the custom oracle changes from the original Venus Protocol Oracle are outlined in the following section.

2.1.4.1. PythOracle

- **Enhanced Precision**: Increases decimal precision from `18` to `30`.
- **Stored Price Fallback**: Stores token prices and applies them if the underlying PythOracle returns a response of `0`.

2.1.4.2. AproOracle

- **Modified Precision**: Based on `Chainlink.sol`, with decimal precision adjusted from `18` to `30`.

2.1.4.3. ResilientOracle

- **Maximization Option**: Adds a `_maximise` parameter, allowing users to select the maximum/minimum price between the MainOracle and FallbackOracle.

2.1.5. Additional Support Contracts

Beyond changes in the Vault and Oracle systems, the Satoshi Preps includes supplementary contracts that enhance logic support and user experience:

2.1.5.1. PythWrapper

- **Oracle Interaction Wrapper**: A wrapper contract to interact with PythOracle, enabling token price retrieval and triggering price update mechanisms.

2.1.5.2. USDGBTC, USDGCORE, USDGUSD Contracts

- **Token Clones**: Replicates GMX's USDG token functionality, providing cloned versions of USDG for specific token applications.

2.2. Findings

During the audit process, the audit team found no vulnerability in the given version of Satoshi Preps.

2.2.1. Fallback to Stored Price Feed When underlyingPythOracle Returns Data as #0 INFORMATIVE

In the PythOracle contract, if the `underlyingPythOracle` returns a data response of #0, the contract defaults to using the stored price to calculate the token's value. However, if the `storedPythPrices` data is outdated, this can lead to an incorrect token price calculation, potentially impacting users' positions.

```
function _getPriceInternal(address asset) internal view returns (uint256 price) {
        TokenConfig storage tokenConfig = tokenConfigs[asset];
        if (tokenConfig.asset == address(0)) revert("asset doesn't exist");

        // if the price is expired after it's compared against `maxStalePeriod`, the
following call will revert
        PythStructs.Price memory priceInfo = underlyingPythOracle.getPriceNoOlderThan(
            tokenConfig.pythId,
            tokenConfig.maxStalePeriod
        );

        if (priceInfo.price == 0) {
            // Use stored Pyth price if current price is zero
            priceInfo = storedPythPrices[asset];  //when the underlyingPythOracle
responding data is equal `#0` then use stored price
            if (priceInfo.price == 0) revert("invalid pyth oracle price");
        }

        uint256 pricePrecision = PRICE_PRECISION; // 1e30
        uint256 basePrice = int256(priceInfo.price).toUint256();
```

```
        if (priceInfo.expo > 0) {
            price = basePrice * (10 ** uint256(int256(priceInfo.expo))) * pricePrecision;
        } else {
            price = (basePrice * pricePrecision) / (10 ** uint256(-
int256(priceInfo.expo)));
        }
    }
```

RECOMMENDATION

Implement a transaction reversion mechanism if the stored price data is outdated.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Oct 29, 2024*	Public Report	Verichains Lab

Table 2. Report versions history