



Smart Contract Code Review And Security Analysis Report

CUSTOMER Bitburo
SCOPE Rebalancer Contract & Keeper
DATE 06.10.2026

Made in Germany by softstack GmbH

Table of Contents

1. Disclaimer	3
2. About the Project and Company	4
3. Vulnerability & Risk Level	5
4. Auditing Strategy and Techniques Applied	6
5. Metrics	7
5.1 Tested Contract Files	7
5.2 Source Lines & Risk	7
5.3 Source Units in Scope	7
6. Scope of Work	8
6.1 Findings Overview	8
6.2 Manual and Automated Vulnerability Test	9
6.2.1 Chainlink round data is not fully validated: answeredInRound/roundId are ignored for both feeds	9
6.2.2 Keeper: a persistent out-of-gas failure is classified as transient ("skipped") and never enters the cool-down ladder; a send exception aborts the remaining owners of the run	11
6.2.3 A Circle blacklist of the Rebalancer contract itself permanently halts all runs and freezes collected fees	13
6.2.4 Availability under oracle/pool divergence is intentionally limited (reverts, never mispriced), verified harmless	15
7. Executive Summary	17
8. About the Auditor	18
9. Glossary	19

1. Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of the client. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Version / Date	Description
0.1 (24.09.2026)	Layout
0.4 (26.09.2026)	Automated Security Testing / Manual Security Testing
0.9 (29.09.2026)	Findings delivered
1.0 (06.10.2026)	Re-check after mitigation and final document

2. About the Project and Company

Company: Pinhead B.V.

Address: Postbus 80, 5600 AB Eindhoven, The Netherlands

Website: <https://bitburo.nl>

GitHub: <https://github.com/pinhead-io>

2.1 Project Overview

Bitburo, developed by Pinhead B.V., is a web application that keeps a user's savings at a fixed split between Bitcoin and US dollars, currently 80% Bitcoin and 20% dollars. Bitburo automatically rebalances the wallet back to that split whenever market movements push it too far away, at most once a day.

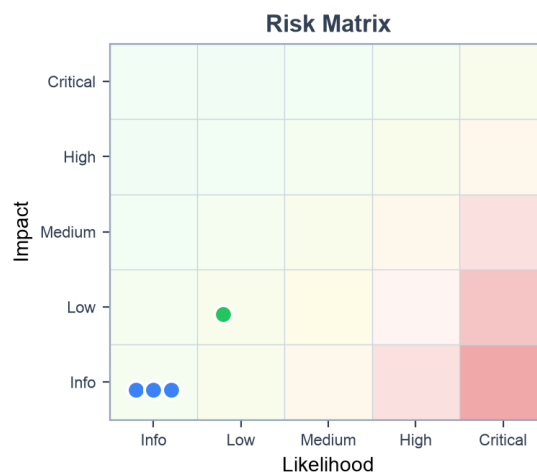
Bitburo is non-custodial: the funds never leave the user's own wallet except for the moment of a trade, and Bitburo cannot move them on its own. The rebalancing rules are fixed in a smart contract on Base that cannot be changed after deployment, so users can rely on the contract behaving exactly as published. Users stay in control at all times and can stop the service or withdraw their permission whenever they want.

The aim is to give everyday users a simple, transparent way to hold a balanced Bitcoin and dollar position, without handing over their funds or having to trade manually.

3. Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.1.

Level	CVSS Score	Vulnerability	Required Action
CRITICAL	9.0 – 10.0	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
HIGH	7.0 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
MEDIUM	4.0 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
LOW	0.1 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
INFORMATIONAL	0.0	A vulnerability that has informational character but is not affecting any of the code.	An observation that does not determine a level of risk.



4. Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

The auditing process follows a routine series of steps:

Code review that includes the following:

- Review of the specifications, sources, and instructions provided to softstack to make sure we understand the size, scope, and functionality of the smart contract.
- Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
- Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to softstack describe.

Testing and automated analysis that includes the following:

- Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
- Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.

Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

5. Metrics

The metrics section gives the reader an overview of the size, quality, flows and capabilities of the codebase.

5.1 Tested Contract Files

Source: https://github.com/pinhead-io/bitburo_contracts
Commit: b5ed2cb2b42add6e6e4cbda0a8a7d6d1a52abce8

File	Fingerprint (MD5)
------	-------------------

5.2 Source Lines & Risk

5.3 Source Units in Scope

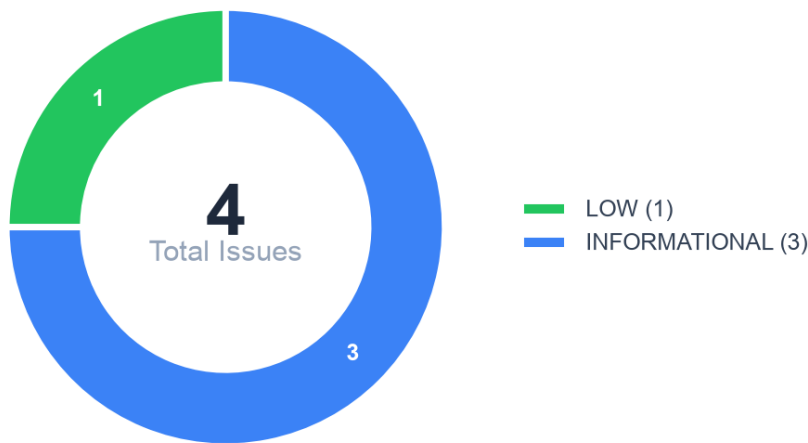
File	Instructions	Functions	Lines	nSLOC	Comments
Total	0	0	0	0	0

6. Scope of Work

This audit covers Bitburo's immutable Rebalancer contract on Base, the deploy script that fixes its immutables, and the off-chain keeper script that runs it. The review focuses on the rebalancing rule and its rounding, the Chainlink price and sequencer-uptime validation, the two-sided price bound around the Uniswap v3 swap, the owner/keeper authorization model, fee handling and the retire kill switch, as well as the keeper's failure classification and cool-down logic.

6.1 Findings Overview

Findings are classified by severity: Critical, High, Medium, Low, and Informational.



No	Title	Severity	CVSS	Status
6.2.1	Chainlink round data is not fully validated: answeredInRound/roundId are ignored for both feeds	LOW	3.7	PARTIALLY FIXED
6.2.2	Keeper: a persistent out-of-gas failure is classified as transient ("skipped") and never enters the cool-down ladder; a send exception aborts the remaining owners of the run	INFORMATIONAL	0.0	PARTIALLY FIXED
6.2.3		INFORMATIONAL	0.0	ACKNOWLEDGED

No	Title	Severity	CVSS	Status
	A Circle blacklist of the Rebalancer contract itself permanently halts all runs and freezes collected fees			
6.2.4	Availability under oracle/pool divergence is intentionally limited (reverts, never mispriced), verified harmless	INFORMATIONAL	0.0	ACKNOWLEDGED

6.2 Manual and Automated Vulnerability Test

6.2.1 Chainlink round data is not fully validated: answeredInRound/roundId are ignored for both feeds

LOW

CVSS 3.7

PARTIALLY FIXED

DESCRIPTION

`_quote()` is the single price authority for every run. It discards `roundId` and `answeredInRound` from both `latestRoundData()` calls and validates only `answer` and `updatedAt` (and, for the sequencer, `startedAt`). Chainlink's documented consumer validation for aggregation-layer anomalies is `updatedAt != 0 && answeredInRound >= roundId`. Two concrete gaps:

Price feed (FEED_CBBTC_USD): a round with `answeredInRound < roundId` (a round the current aggregator never answered, the historical anomaly this check exists for) is accepted as long as `answer > 0` and `updatedAt` is fresh.

Sequencer feed (SEQUENCER_FEED): the uptime round itself is never freshness-checked. If the uptime feed stops transmitting while its last answer is 0 ("up"), `upSince` stays frozen and the sequencer gate passes forever; halt detection then rests entirely on the price feed's `MAX_FEED_AGE` check, which only trips once price updates also stop (more than 25 minutes into an outage).

Impact is bounded: any accepted round is still at most 1500 s old, and the $\pm 0.55\%$ both-side execution bound means a bad price halts trading rather than mispricing a run. A stale or deprecated feed returning zeros is already caught by `InvalidFeedAnswer`. Severity is therefore Low rather than Medium.

CODE LOCATION

`src/Rebalancer.sol`, function `_quote` (lines 260–269 at audit time)

```
(, int256 sequencerAnswer, uint256 upSince,,) =
SEQUENCER_FEED.latestRoundData();
// upSince 0 is an uninitialized round and counts as down, like a time in
the future.
if (
```

```

        sequencerAnswer != 0 || upSince == 0 || upSince > block.timestamp
        || block.timestamp - upSince < GRACE_PERIOD
    ) revert SequencerDown();

    (, int256 answer,, uint256 updatedAt,) = FEED_CBBTC_USD.latestRoundData();
    if (answer <= 0) revert InvalidFeedAnswer();
    if (updatedAt > block.timestamp || block.timestamp - updatedAt >
MAX_FEED_AGE) revert StaleFeed();

```

```

contract BrokenRoundAggregator is IAggregatorV3 {
    uint80 public roundId = 1;
    int256 public answer;
    uint256 public startedAt;
    uint256 public updatedAt;
    uint80 public answered;

    function set(int256 answer_, uint256 startedAt_, uint256 updatedAt_, uint80
answeredInRound_) external {
        roundId++;
        answer = answer_;
        startedAt = startedAt_;
        updatedAt = updatedAt_;
        answered = answeredInRound_;
    }

    function decimals() external pure returns (uint8) {
        return 8;
    }

    function latestRoundData() external view returns (uint80, int256, uint256,
uint256, uint80) {
        return (roundId, answer, startedAt, updatedAt, answered);
    }
}

contract AuditProbe is RebalancerTestBase {
    function test_probe_answeredInRoundIgnored() public {
        vm.warp(1_800_000_000);
        BrokenRoundAggregator bad = new BrokenRoundAggregator();
        // Fresh, valid answer, but answeredInRound (0) < roundId (2): accepted
        anyway.
        bad.set(int256(80_000e8), vm.getBlockTimestamp(), vm.getBlockTimestamp(),
0);

        Rebalancer r2 = new Rebalancer(
            address(usdc), address(cbbtc), address(router), address(bad),
            address(sequencer), treasury
        );
    }
}

```

```

        assertEq(r2.quote(), 80_000e6, "answeredInRound < roundId is accepted");
    }

    function test_probe_sequencerRoundNeverCheckedForStaleness() public {
        // Uptime feed last touched a year ago with answer 0: the gate passes for
        ever.
        sequencer.set(0, vm.getBlockTimestamp() - 365 days, vm.getBlockTimestamp()
        - 365 days);
        assertEq(r.quote(), PRICE, "a year-old up-round passes");
    }
}

```

RECOMMENDATION

Complete the round validation per Chainlink's consumer guidelines. For the price feed, capture the full tuple (uint80 roundId, int256 answer,, uint256 updatedAt, uint80 answeredInRound) and add if (answeredInRound < roundId) revert InvalidFeedAnswer();. For the sequencer feed, additionally require its round to be non-stale (for example its own updatedAt within a generous window, longer than typical "up" periods so the feed's change-only update cadence is respected). The contract is immutable, so the fix ships with the next deployment profile; given the bounded impact, deferral is justifiable.

MITIGATION

https://github.com/pinhead-io/bitburo_contracts/commit/d8e9df9

Partially fixed / acknowledged. Sequencer feed: fixed. _quote() now also requires the uptime round to have been updated within MAX_SEQUENCER_FEED_AGE (3 days) and reverts with StaleFeed otherwise. On Base the uptime feed is refreshed once a day (sampled over the last 30 days), so 3 days tolerates two missed refreshes. The answeredInRound >= roundId check was not added: Chainlink's API reference marks answeredInRound as "Deprecated - Previously used when answers could take multiple rounds to be computed", and Bitburo prefers not to make an immutable contract depend on a deprecated field. The checks on answer > 0 and on the age of the answer (1500 s) remain. softstack verified the sequencer staleness check at tag audit-2026-10-01 and considers the remaining part an acceptable, documented decision.

6.2.2 Keeper: a persistent out-of-gas failure is classified as transient ("skipped") and never enters the cool-down ladder; a send exception aborts the remaining owners of the run

INFORMATIONAL

CVSS 0.0

PARTIALLY FIXED

DESCRIPTION

The keeper sends execute with a fixed GAS_LIMIT (default 500,000). If a transaction reverts on chain because it ran out of that fixed gas (not a contract revert), the post-mortem re-simulation runs via eth_call, which applies no gas cap, so it succeeds; the run is classified "skipped", the cool-down counter is never incremented, and the same owner is retried (paying a real transaction) every keeper cycle, forever, instead of backing off exponentially as designed.

Separately, any exception thrown by writeContract / waitForTransactionReceipt (e.g. insufficient ETH) propagates out of the for loop and abandons the remaining due owners for that cycle. This is harmless given the 1-day on-chain interval and the 10–60 minute keeper cadence, but only the already-processed owners' cool-down state is preserved via the finally write.

Measured on a Base fork at the deploy block, execute costs ~202k gas (buy) / ~212k (sell), a ~2.4× margin under the default limit, so the out-of-gas regime requires an anomalous future state.

CODE LOCATION

keeper/keeper.ts, function run (lines 316–340 at audit time); off-chain

```
const hash = await walletClient.writeContract({
  address: rebalancer,
  abi,
  functionName: "execute",
  args: [owner, demanded],
  gas: gasLimit,
});

const receipt = await publicClient.waitForTransactionReceipt({ hash, timeout:
txWaitSeconds * 1000 });

if (receipt.status === "success") return { outcome: "executed", detail: `${
{label}: ${hash}` };

// Reverted on chain. Ask the node why, on the state after that block: usually
the price moved past `minAmountOut`.
try {
  await publicClient.simulateContract({
    address: rebalancer,
    abi,
    functionName: "execute",
    args: [owner, demanded],
    account: keeper,
    blockNumber: receipt.blockNumber,
  });

  return { outcome: "skipped", detail: `${{label}: ${hash} reverted, a new
simulation passes; next run` };
}
```

```
# Run against the keeper as it was before commit da93ad3; the steps below show the
# behaviour that fix removes.
cd keeper && bun install
```

```

RPC_URL=https://mainnet.base.org
REBALANCER=0x0C83C3785deF69B25c08b190C607D5890dfACE66 \
FROM_BLOCK=52034272 GAS_LIMIT=150000 DRY_RUN=0 KEEPER_PRIVATE_KEY=0x... bun run
keeper.ts
# Every send for a due owner reverts out-of-gas; each cycle logs
#   "<owner>: skipped: <hash> reverted, a new simulation passes; next run"
# and the cool-down file never gains an entry for that owner: the owner is retried
# (one wasted transaction per cycle) on every subsequent run, indefinitely.

```

RECOMMENDATION

Record a cool-down when a transaction has already failed for the same owner repeatedly (for example treat "reverted receipt + passing re-simulation" as a one-strike event after N consecutive occurrences), and wrap the per-owner run() call in a try/catch inside the loop so one owner's send exception does not abandon the rest of the cycle. Optionally compare receipt.gasUsed against gasLimit to detect the out-of-gas class directly.

MITIGATION

https://github.com/pinhead-io/bitburo_contracts/commit/da93ad3

Partially fixed / acknowledged. Out of gas: fixed. When a transaction reverted and the re-simulation afterwards passes, the keeper now simulates once more with the gas limit of the transaction; if that fails, the owner is put on the cool-down. Bitburo reproduced this on a local fork with GAS_LIMIT=150000: one transaction, then cool-down. A send exception that ends the cycle: acknowledged and left as it is on purpose. A send that fails for one owner (no ETH, for example) fails for the following owners too, and the script runs again within the hour while the contract allows one run per owner per day. softstack verified the out-of-gas fix in the keeper at tag audit-2026-10-01: a reverted receipt whose re-simulation passes is now re-simulated once more with the gas limit of the transaction, and a failure of that second simulation returns the cool-down outcome instead of "skipped". The remaining part is an acceptable, documented decision.

6.2.3 A Circle blacklist of the Rebalancer contract itself permanently halts all runs and freezes collected fees

INFORMATIONAL

CVSS 0.0

ACKNOWLEDGED

DESCRIPTION

USDC on Base is a blacklistable FiatToken. The run flow pulls both legs through the contract (safeTransferFrom(owner, address(this), ...) for the input; router payout to the contract for the output), so if Circle blacklists the Rebalancer address, every execute and every sweep reverts; collected fees are frozen for as long as the blacklist holds, and retire() provides no recovery.

User funds are not at risk: they remain in user wallets and the unusable allowance can simply be revoked. This is a structural risk of integrating blacklistable stablecoins; at audit time the README considered the blacklist only for the treasury.

CODE LOCATION

src/Rebalancer.sol, functions execute and sweep (line 226, sell leg at lines 233 and 244–246 at audit time)

```
USDC.safeTransferFrom(owner, address(this), amountIn);

function sweep(IERC20 token) external nonReentrant {
    token.safeTransfer(TREASURY, token.balanceOf(address(this)));
}
```

```
function test_probe_contractBlacklistedHaltsRunsAndFreezesFees() public {
    _setUser(alice, 1000e6, 0, 8000);
    _setUser(bob, 0, 0.0125e8, 5000);
    vm.prank(keeper);
    r.execute(alice, 0); // leaves a 0.25 USDC fee in the contract
    usdc.setBlacklisted(address(r), true);
    vm.prank(keeper);
    vm.expectRevert("Blacklistable: account is blacklisted");
    r.execute(bob, 0); // the pull from the owner into the blacklisted
contract fails
    vm.expectRevert("Blacklistable: account is blacklisted");
    r.sweep(usdc); // the fees are frozen as long as the blacklist holds
    assertEq(usdc.balanceOf(address(r)), 250_000, "fee still in the
contract");
    assertEq(cbbtc.balanceOf(bob), 0.0125e8, "user funds untouched");
}
```

RECOMMENDATION

Nothing to change on-chain; the design already isolates user funds. Operationally, monitor Circle blacklist events involving the contract and treat the contract address's blacklistability as a known shutdown condition; if fee exposure grows, sweep more frequently so the frozen-fee surface stays small.

MITIGATION

Acknowledged and documented. No code change. The README (section Fee) now describes what happens when Circle blacklists the contract itself: every run and every sweep reverts and unswept fees are frozen for as long as the blacklist holds, while user funds remain in the users' own wallets.

6.2.4 Availability under oracle/pool divergence is intentionally limited (reverts, never mispriced), verified harmless

INFORMATIONAL

CVSS 0.0

ACKNOWLEDGED

DESCRIPTION

Three documented availability edges were re-verified by this audit and confirmed to be revert-only, never lossy:

MAX_FEED_AGE of 1500 s allows a feed up to 25 minutes stale, which in a fast market can exceed the $\pm 0.55\%$ execution bound, so runs revert until the feed catches up (heartbeat 20 min / 0.3% deviation threshold).

The upper bound blocks "too good" fills (the stuck-feed / USDC-depeg direction), which is correct fail-stop behavior.

Above $\sim 100,000$ USDC per cbBTC, one satoshi exceeds 5 bps of a minimum-size run, so such a run can revert on a pool ~ 50 bps off the feed and is retried later.

In every probed case the failure is atomic: nothing moved, no interval started.

CODE LOCATION

src/Rebalancer.sol, constants MAX_DEVIATION_BPS, GRACE_PERIOD, MAX_FEED_AGE (lines 76–78)

```
uint256 public constant MAX_DEVIATION_BPS = 55;
uint256 public constant GRACE_PERIOD = 1 hours; // after the sequencer comes
back up
uint256 public constant MAX_FEED_AGE = 1500; // cbBTC/USD heartbeat of 20
minutes + 25%
```

```
function testFuzz_probe_boundsAtExtremePrices(uint256 cash, uint256 btc,
uint16 target, int256 worseBps, uint256 price)
    public
    {
        price = bound(price, 1e6, 1e12); // $1 .. $1,000,000 per cbBTC
        _setPrice(price);
        cash = bound(cash, 0, 1e13);
        btc = bound(btc, 0, 1e11);
        target = uint16(bound(target, 500, 9500));
        worseBps = bound(worseBps, -55, 54);
        _setUser(alice, cash, btc, target);
        router.setWorseBps(worseBps);
        (bool due,) = r.isDue(alice);
        vm.assume(due);
        uint256 valueBefore = _valueOf(alice);
        uint256 usdcBefore = usdc.balanceOf(alice);
        uint256 btcBefore = cbbtc.balanceOf(alice);
```

```
vm.prank(keeper);
try r.execute(alice, 0) returns (uint256) {
    // on success: loss <= 80 bps of run + 3 sats, MIN_TRADE <= run <= cap
    + MIN + 1 sat,
    //                a run beyond the cap leaves nothing unfinished
} catch {
    // on revert: balances unchanged, interval not started
}
}
```

RECOMMENDATION

None required: the fail-stop behavior is the intended safety property. For a future deployment profile, consider shrinking MAX_FEED_AGE toward the 1200 s heartbeat if availability during volatility matters more than tolerance for late transmitters.

MITIGATION

Acknowledged, no change. MAX_FEED_AGE stays at 1500 s: the 20-minute cbBTC/USD heartbeat plus 25% for a transmitter that is late.

7. Executive Summary

Two independent softstack experts performed an unbiased and isolated audit of the smart contract and keeper script provided by the Bitburo team. The main objective of the audit was to verify the security and functionality of the immutable Rebalancer contract on Base, which rebalances a user's wallet between USDC and cbBTC through a revocable ERC-20 allowance. The audit process involved a thorough manual code review and automated security testing.

The audit identified a total of 4 issues, classified as follows:

- No critical issues were found
- No high severity issues were found
- No medium severity issues were found
- 1 low severity issue was found
- 3 informational issues were identified

None of the findings put user funds at risk. The contract keeps funds in the users' own wallets, bounds every swap on both sides against Chainlink, and fails atomically when its preconditions are not met.

Update (06.10.2026): Bitburo addressed the findings at tag audit-2026-10-01, which is the code of the live contract on Base at 0x0C83C3785deF69B25c08b190C607D5890dfACE66 (verified on Basescan). L-01 was partially fixed and acknowledged: the sequencer uptime round is now freshness-checked (MAX_SEQUENCER_FEED_AGE of 3 days), while the answeredInRound check was deliberately not added because Chainlink marks the field as deprecated. I-01 was partially fixed and acknowledged: a persistent out-of-gas failure now puts the owner on the cool-down, while a send exception still ends the cycle by design. I-02 was acknowledged and documented in the README, and I-03 was acknowledged without change. At the tag, Bitburo reports 83 tests passing without a fork, 93 with the fork suite at the deploy block, 14 deep invariants and 42 keeper tests. No further concerns remain for this release.

8. About the Auditor

Established in 2017 under the name Chainsulting, and rebranded as softstack GmbH in 2023, softstack has been a trusted name in Web3 Security space. Within the rapidly growing Web3 industry, softstack provides a comprehensive range of offerings that include software development, cybersecurity, and consulting services. Softstack's competency extends across the security landscape of prominent blockchains like Solana, Tezos, TON, Ethereum and Polygon. The company is widely recognized for conducting thorough code audits aimed at mitigating risk and promoting transparency.

The firm's proficiency lies particularly in assessing and fortifying smart contracts of leading DeFi projects, a testament to their commitment to maintaining the integrity of these innovative financial platforms. To date, softstack plays a crucial role in safeguarding over \$100 billion worth of user funds in various DeFi protocols.

Underpinned by a team of industry veterans possessing robust technical knowledge in the Web3 domain, softstack offers industry-leading smart contract audit services. Committed to evolving with their clients' ever-changing business needs, softstack's approach is as dynamic and innovative as the industry it serves.

Check our website for further information: <https://softstack.io>

9. Glossary

Term	Definition
BPS	Basis points; 1 bp = 0.01%
Base	Coinbase-incubated Ethereum Layer-2 (OP Stack) on which the Rebalancer is deployed
Bitburo	Pinhead B.V.'s web application that keeps a user's wallet at a fixed BTC/USD split
CVSS	Common Vulnerability Scoring System
Chainlink	Decentralized oracle network providing the cbBTC/USD price feed
ERC-20	Ethereum-compatible fungible token standard
Heartbeat	Maximum interval after which a Chainlink feed publishes a new round even without a price change
Keeper	Off-chain script, or the address running it, that an owner appoints to call execute on their behalf
Mandate	A user's on-chain instruction to the Rebalancer, consisting of the target cbBTC share and the appointed keeper
Multicall3	Contract that batches several read calls into a single RPC request
OZ	OpenZeppelin, widely-used Solidity library for secure smart contracts
Privy	Wallet infrastructure provider used for Bitburo's embedded user wallets and the keeper's server wallet
Rebalancer	Bitburo's immutable contract that rebalances a wallet between USDC and cbBTC via an ERC-20 allowance
Sequencer Uptime Feed	Chainlink feed that reports whether the Base L2 sequencer is up (answer 0) or down
SwapRouter02	Uniswap router contract; the Rebalancer only calls exactInputSingle
USDC	Circle's USD stablecoin; a blacklistable FiatToken on Base
Uniswap v3	Concentrated-liquidity AMM whose USDC/cbBTC 0.05% pool the Rebalancer swaps through
cbBTC	Coinbase Wrapped BTC, an ERC-20 token backed 1:1 by Bitcoin