Botanix stBTC Security Review

Auditors

Chris Smith, Lead Security Researcher

Noah Marconi, Lead Security Researcher

Sujith somraaj, Security Researcher

Report prepared by: Lucas Goiriz

June 17, 2025

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Botanix is building a new Layer 2 protocol called the Spiderchain that supports a decentralized financial system running on Bitcoin.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of Botanix stBTC according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 1 days in total, Botanix engaged with Spearbit to review the botanix-stBTC protocol. In this period of time a total of **16** issues were found.

Summary

Project Name	Botanix
Repository	botanix-stBTC
Commit	0dcfbac5
Type of Project	L2, Staking
Audit Timeline	Apr 16th to Apr 17th

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	1	1	0
Medium Risk	0	0	0
Low Risk	1	1	0
Gas Optimizations	0	0	0
Informational	14	9	5
Total	**16**	**11**	**5**

5 Findings

5.1 High Risk

5.1.1 Unprotected `stBTC.notifyRewardAmount` allows griefing to DoS significant reward amounts

Severity: High Risk

Context: stBTC.sol#L137

Description: `stBTC.notifyRewardAmount` is an unprotected function meaning it can be called by anyone, at any time. Within the function there is rounding that occurs when determining the reward rate.

`rewardRate = rewardBalance / REWARDS_DURATION;` rounds down by the amount `rewardBalance % REWARDS_DURATION`. E.g. in the extreme case a reward amount of 1209599, virtually half the rewards would be excluded from the reward rate `(REWARDS_DURATION + REWARDS_DURATION - 1) / REWARDS_DURATION = 1`.

A malicious actor may exploit this scenario by calling `stBTC.notifyRewardAmount` each block. The proof of concept below shows how 1 BTC worth of rewards is reduced considerably over 7 days by calling every 5 seconds.

Proof of Concept: Add to `test/sBTC.t.sol`:

```
function testMaliciousNotify() public {
    // 1. Alice deposits pBTC into the sBTC vault
    vm.startPrank(alice);
    uint256 aliceBalanceBefore = pBTC.balanceOf(alice);
    uint256 depositShares = sbtc.deposit(INITIAL_DEPOSIT, alice);
    vm.stopPrank();

    // 2. Send rewards to FeeReceiver and distribute
    vm.deal(address(feeTo), 1 ether);
    feeTo.harvest();

    for (uint256 i = 0; i < 7 days; i += 5) {
        // next block in seconds.
        skip(5);
        sbtc.notifyRewardAmount();
    }

    // Move all rewards to stake
    sbtc.harvest();

    assertEq(sbtc.totalAssets(), INITIAL_DEPOSIT + 1 ether, "Total assets include deposits and all
    ↪  emitted rewards");
}
```

Recommendation: Limit how often `notifyRewardAmount` may be called and consider adding handling of `rewardBalance % REWARDS_DURATION`.

Botanix: Fixed in commit fa0411bf.

Spearbit: The `minHarvestInterval` removes the DoS and leaves only minimal rounded amounts to emit later. Moving `lastHarvest = block.timestamp;` to L68 would make maintain checks/effects/interactions ordering. For duration, over a 7 day period of emissions, the most delayed amount (by notifying every 2 days) is:

```
 [0] VM::assertEq(10099999999999734400 [1.009e20], 10100000000000000000 [1.01e20], "Total assets
↪  include deposits and all emitted rewards") [staticcall]
    ← [Revert] Total assets include deposits and all emitted rewards: 10099999999999734400 !=
↪  10100000000000000000
 ← [Revert] Total assets include deposits and all emitted rewards: 10099999999999734400 !=
↪  10100000000000000000
```

You can wargame scenarios by adding this test to `sBTCTest` and modifying the variable for `minHarvestInterval`.

```
function testMaliciousNotify() public {
    // 1. Alice deposits pBTC into the sBTC vault
    vm.startPrank(alice);
    uint256 aliceBalanceBefore = pBTC.balanceOf(alice);
    uint256 depositShares = sbtc.deposit(INITIAL_DEPOSIT, alice);
    vm.stopPrank();

    // 2. Send rewards to FeeReceiver and distribute
    // First, send native tokens to FeeReceiver
    vm.deal(address(feeTo), 1 ether);

    feeTo.harvest();

    uint256 minHarvestInterval = 2 days;
    for (uint256 i = 0; i < 7 days; i += minHarvestInterval) {
        skip(minHarvestInterval);
        sbtc.notifyRewardAmount();
    }

    // Move all rewards to stake
    skip(7 days);
    sbtc.harvest();

    assertEq(sbtc.totalAssets(), INITIAL_DEPOSIT + 1 ether, "Total assets include deposits and all
    ↪  emitted rewards");
}
```

5.2 Low Risk

5.2.1 Permanent locking of native tokens in `directDeposit()` if `msg.value` and `assets` don't match

Severity: Low Risk

Context: stBTC.sol#L228

Description: The `directDeposit()` function in `stBTC.sol` enables users to make deposits using native tokens rather than the vault asset. Behind the scenes, the function wraps the native tokens sent into pegged bitcoin (the vault asset). However, a problem arises in this function when the user submits an **asset** parameter that does not match the **msg.value** amount while having a sufficient allowance for the contract to transfer BTC tokens:

```
if (msg.value > 0 && msg.value == assets) { /// <--- if the msg.value != assets, then it enters the else
↪  case
   PeggedBitcoin(payable(address(asset()))).deposit{ value: msg.value }();
} else {
    SafeTransferLib.safeTransferFrom(sERC20(asset()), caller, address(this), assets);
}
```

The issue occurs in the conditional branch. If `msg.value > 0` but `msg.value != assets`, the function will skip converting the native tokens to pBTC and instead attempt to transfer pBTC tokens from the user. The sent native tokens become trapped in the contract as there's no mechanism to return them.

Proof of Concept: Place the following test in the `sBTC.t.sol` file under the `/test` folder:

```
function test_lockingOfNativeTokens() public {
    vm.startPrank(bob);
    sbtc.directDeposit{value: 1.1 ether}(1 ether ,bob);

    assert(address(sbtc).balance == 1.1 ether);
}
```

The proof of concept mentioned above clarifies that rather than refunding the native tokens, the function locks them in the `stBTC.sol` contract and utilizes the user's pBTC balance to finalize the direct deposit.

Recommendation: Consider validating the `msg.value` against the `assets` parameter to prevent the function from executing when there's a mismatch:

```
function _deposit(address caller, address receiver, uint256 assets, uint256 shares) internal override {
    // Update our elastic (non-shares) value to track balance
    totalStaked += assets;

    // Perform the underlying deposit
    if (msg.value > 0) {
        require(msg.value == assets, "ETH value must match assets");
        // If the user sent ETH, deposit it to the asset token
        PeggedBitcoin(payable(address(asset()))).deposit{ value: msg.value }();
    } else {
        // Otherwise, transfer the assets from the caller to this contract
        SafeTransferLib.safeTransferFrom(sERC20(asset()), caller, address(this), assets);
    }

    _mint(receiver, shares);

    emit Deposit(caller, receiver, assets, shares);
}
```

Botanix: Fixed in commit 035d048e.

Spearbit: Fix verified.

5.3 Informational

5.3.1 Native token terminology mismatch

Severity: Informational

Context: stBTC.sol#L229

Description: The `stBTC.sol` contract is designed to work on a Bitcoin L2 blockchain where the native token is Bitcoin, yet the code comments consistently refer to the native token as "ETH." This inconsistency might confuse developers, auditors, and users engaging with the contract.

Recommendation: Update all instances of "ETH" in comments to refer to the native Bitcoin token used on this L2 chain.

Botanix: Fixed in commit 9dc673f9.

Spearbit: Fix verified.

5.3.2 Remove unused file imports

Severity: Informational

Context: stBTC.sol#L11

Description: The `stBTC.sol` imports the `ERC20Upgradeable.sol` contract from the `openzeppelin-contracts-upgradeable` library, but the file is not referenced or used anywhere in the code.

Recommendation: Consider removing the above-mentioned unused file import.

Botanix: Fixed in commit 644d963e.

Spearbit: Fix verified.

5.3.3 Redundant `SafeERC20` library import

Severity: Informational

Context: stBTC.sol#L31

Description: The `stBTC.sol` contract imports the `SafeERC20` library from OpenZeppelin and declares its usage through a `using for` statement but never actually utilizes it in the code. Instead, the contract uses Solmate's `SafeTransferLib` for token transfer operations.

```
import { SafeERC20 } from "openzeppelin-contracts-upgradeable/lib/openzeppelin-contracts/contracts/toke
↪  n/ERC20/utils/SafeERC20.sol";
// ...
contract stBTC is Initializable, Ownable2StepUpgradeable, ERC4626Upgradeable {
   // ...
   using SafeERC20 for ERC20;
   // ...
   SafeTransferLib.safeTransferFrom(sERC20(asset()), caller, address(this), assets);
}
```

Recommendation: Remove the redundant import and **using** statement to improve code clarity.

```
- import { ERC20 } from "openzeppelin-contracts/contracts/token/ERC20/ERC20.sol";
  // ...
- import { SafeERC20 } from "openzeppelin-contracts-upgradeable/lib/openzeppelin-contracts/contracts/to
↪  ken/ERC20/utils/SafeERC20.sol";
  // ...
- using SafeERC20 for ERC20;
```

Botanix: Fixed in commits 76b7c500 and c9d1f2bf.

Spearbit: Fix verified.

5.3.4 Missing parameter validation in `initialize()` functions

Severity: Informational

Context: FeeReceiver.sol#L33, stBTC.sol#L118

Description: The `FeeReceiver.sol` and `stBTC.sol` contracts `initialize()` function lack sanity validations for its input parameters, potentially allowing the contract to be permanently lost with invalid addresses.

Recommendation: Consider adding proper input validations in the `initialize()` functions as follows:

```
  // contract stBTC.sol

  function initialize(IERC20 _asset, address _initialOwner) external initializer {
+     require(address(_asset) != address(0), "invalid asset");
+     require(_initialOwner != address(0), "invalid owner");
      __Ownable_init(_initialOwner);
      __ERC20_init("Staked Bitcoin", "stBTC");
      __ERC4626_init(_asset);
  }

  // contract FeeReceiver.sol

  function initialize(PeggedBitcoin _pbtc, address _stakingVault) external initializer {
+    require(address(_pbtc) != address(0), "invalid _pbtc");
+    require(_stakingVault != address(0), "invalid vault");
     pBTC = _pbtc;
     stakingVault = _stakingVault;
  }
```

Botanix: Fixed in commits c6c49def and 76b7c500.

Spearbit: Fix verified.

5.3.5 Move storage gap variable after all state variable declarations

Severity: Informational

Context: stBTC.sol#L65

Description: In upgradeable smart contracts, using a storage gap is a general practice to reserve slots for future variable additions while maintaining compatibility with previous versions.

The `stBTC.sol` contract declares a storage gap (`uint256[50] private ______gap;`) but places it before other state variables (periodFinish, rewardRate, totalStaked, etc...), deviating from the standard implementation of storage gaps, which usually is placed after all state variable declarations.

This possibly leads to the following scenario (assuming the first 50 slots are reserved for future variables):

1. Placing a variable before the `______gap`: In this case, slot 0 will be allocated to the new variable, and the `_____gap` will be reserved from slots 1 - 49, where 50 will be `periodFinish` and so on.
2. Placing a variable after the `______gap`: In this case, slot 49 will be allocated to the new variable and the `_____gap` will be reserved from slots 0 - 48, where 50 will be `periodFinish` and so on.
3. Placing a variable after the `totalStaked` variable: (with updating `_____gap` size to 49). In this case, storage collision happens, as slot 49 will now be reserved for the `periodFinish` variable instead of 50, and the new variable will be added to slot 56, which is reserved for the `totalStaked` variable.

Recommendation:

1. Move the storage gap to the end of the contract's storage variables to ensure reserved slots are positioned after all current variables.

```
contract stBTC is Initializable, Ownable2StepUpgradeable, ERC4626Upgradeable {
    // ...
    uint256 public periodFinish;
    uint256 public rewardRate;
    uint256 public totalStaked;
    // ... (other variables)

    // Reserve storage slots for future upgrades
    uint256[50] private ______gap; // <-- Correct placement at the end
}
```

2. Consider documenting this behavior if the slot is added before variable declarations. Warn that adding new variables at the end may cause data corruption.

Botanix: Fixed in commit df02055e.

Spearbit: Fix verified.

5.3.6 WETH9 Notes

Severity: Informational

Context: pBTC.sol#L49

Description: pBTC is modified from the WETH9 contract. Key differences include:

- The token name and symbol.
- Solidity version change.
- Use of later solidity features such as the `receive` function, checked math, and explicit `type(uint256).max`.

Using `receive` over the fallback function means the `Silent Fallback Method` issue has been eliminated. Checked math does have gas implications but overflows are not anticipated or desired.

Of note, WETH9 behavior is inherited:

- `totalSupply` can be greater than sum of each `balanceOf` due to `selfdestruct` or consensus layer balance updates.
- Integration inefficiencies.

Recommendation: No edits recommended. Consider documenting for integrators to be aware.

Botanix: Acknowledged.

Spearbit: Acknowledged.

5.3.7 Outdated comment referencing fees

Severity: Informational

Context: stBTC.sol#L158

Description: Recommend updating the comment to remove the reference to fees.

Botanix: Fixed in commit b6de6704.

Spearbit: Fix verified.

5.3.8 Asymmetric reward growth leads to minting zero shares

Severity: Informational

Context: stBTC.sol#L223

Description: The `stBTC.sol` contract has an issue with its share calculation logic. This issue causes users to receive zero shares when depositing assets after the contract has accumulated substantial rewards. The `_convertToShares()` function in the `ERC4626Upgradeable.sol` contract from OpenZeppelin is used to calculate the number of **shares ** a user will receive after a successful deposit:

```
function _convertToShares(uint256 assets, Math.Rounding rounding) internal view virtual returns
↪  (uint256) {
   return assets.mulDiv(totalSupply() + 10 ** _decimalsOffset(), totalAssets() + 1, rounding);
}
```

This function works well if the product of `assets` to deposit and the `totalSupply()` is greater than the `totalAssets()`. But in `stBTC.sol`, the `totalAssets()` value can increase over time due to accumulating rewards from the consensus layer without user deposits, leading to the scenario affecting users trying to deposit assets less than the `totalAssets()` value.

This issue can be mitigated by minting initial shares, but this action alone is insufficient. The issue can surface again:

- If the initial minter redeems his shares and `totalSupply()` drops significantly, causing the product of `totalSupply()` and assets to deposit to fall below `totalAssets()`.
- If the contract rewards are too high, inflating the `totalAssets()` will result in minting zero shares for small depositors.

Proof of Concept: The following scenario demonstrates the vulnerability:.

- Alice stakes `1e18`.
- Fee receiver sends out `2e18` tokens in rewards.
- 2 days after fee receiver sends out another `2e18` tokens.
- Alice withdraws her funds like 2 days after claiming some tokens.

- Now if a new user deposits any value < `totalAssets()` in the vault, they receive zero shares (meaning they lose their entire funds).

```
function test_flow() public {
    vm.startPrank(alice);
    sbtc.deposit(2e18, alice);

    /// 1e18 sent as rewards
    vm.deal(address(feeTo), 2e18);
    feeTo.harvest();

    vm.warp(block.timestamp + 2 days);

    vm.deal(address(feeTo), 2e18);
    feeTo.harvest();

    vm.warp(block.timestamp +  1 days);
    sbtc.redeem(sbtc.balanceOf(alice), alice, alice);

    vm.warp(block.timestamp + 7 days);
    console.log("convert to shares:", sbtc.convertToShares(100e18));

    vm.startPrank(bob);
    sbtc.deposit(1e18, bob);
}
```

Recommendation: Consider validating the number of shares in the `_deposit()` function:

```
  function _deposit(address caller, address receiver, uint256 assets, uint256 shares) internal override
  ↪  {
+    require(shares > 0, "insufficient shares");
     // ...
  }
```

Mint enough shares upfront and avoid redeeming them to ensure the `totalSupply()` value is high enough to support depositing smaller asset values.

Botanix: Zero share minting is now fixed as of commit eeb92951.

Spearbit: Fix verified.

5.3.9 Incorrect min deposit protection in Invariant Suite

Severity: Informational

Context: sBTCInvariants.t.sol#L44-L47

Description: This block of code is meant to mimic the expected behavior of the deployed code where Botanix will deposit some amount of pBTC to `address(1)` in order to protect from the known 4626 minimum share vulnerability. However, this code does not work as expected in the invariant suite because `alice` is assigned to `address(0x1)` meaning the `totalSupply` of the invariant code can be 0.

Recommendation: Reassign alice to a different address or use a different address to block full withdraw.

Botanix: Reassigned address in commit cf2ef6ef.

Spearbit: Fix corrects overlapping addresses in invariant tests.

5.3.10 Precision tolerance in Invariant suite is too high

Severity: Informational

Context: sBTCInvariants.t.sol#L288-L289

Description: The invariant suite tests whether the conversion on a test amount to `shares` and back to `assets` will result in a significant deviation between the initial asset amount and the calculated one. However, it allows for a 1% tolerance or 0.01 BTC which does not seem to be economically insignificant and acceptable, especially if the rounding can be replayed to compound the deviation though deposits and withdraws which also rely on the conversion of assets. Further adjusting this tolerance down towards `0.00001e18` or `.0001%` starts to surface failures in the invariant tests. For instance, at that tolerance, this sequence is a regression:

```
withdraw(19463556330487544860944996498242426376,
↪  115792089237316195423570985008687907853269984665640564039457584007913129639935);
addRewards(102725981152976193958644346911775);
addRewards(20);
addRewards(42912040319958687918578654924026686340591476621179463136O);
advanceTime(81531340733012568228756154638047 2);
advanceTime(11297);
advanceTime(1);
addRewards(1820089340640817);
addRewards(154472958145110159646778455653642080347141362005244239732614);
harvest();
advanceTime(4626);
addRewards(230);
advanceTime(863050489767661103540298400544478174392604577369301200948341814494861302169 0);
advanceTime(42912040319958687918578654924026686340591476621179463135 );
deposit(475346554570287, 952274666163953119829687520);
addRewards(29863952071095354306011138689598935);
advanceTime(1630);
deposit(3901, 1950811);
deposit(5081097062443745093, 6492);
deposit(19696458258013583235 , 5543);
deposit(2299, 775);
withdraw(34367408130659740052778147060028088306912996 3, 207381174625319465364723369518872 9);
withdraw(449826859, 4615907);
withdraw(10182, 11911205);
withdraw(2003, 86);
deposit(8250661375661, 16512951785028);
deposit(443762893583409954872, 64781614);
deposit(234678126209833028758744017410063648063024759829143756179781502239012148 9,
↪  119431984859561271468898683018725940709328801266203732955);
harvest();
assertSharesValues();
```

Recommendation: To have greater confidence in the rounding across different conditions, reduce the precision of this test and investigate failures.

One way to "*tighten*" the rounding precision of the protocol is to ensure calculations do not round down more than once. For instance, when performing `notifyRewardAmount`'s update to the `rewardRate`, the math rounds down once in `rewardPerToken` and once in `earned` and a third time in `rewardRate = rewardBalance / REWARDs_DURATION`. This can be solved by "inflating the numbers" to `RAY` or `RAD` precision until then need to be stored so in this case the math in `rewardPerToken` and in `earned` would be performed at a higher precision and only rounded down to `WAD` precision when it was being stored in `rewardRate`. `_convertToShares` and `_convertToAssets` have similar multiple rounds in their math due to `totalAssets()` being calculated dynamically.

Botanix: Acknowledged.

Spearbit: Acknowledged.

5.3.11 Improved invariant testing

Severity: Informational

Context: foundry.toml#L1-L22

Description: Currently, reverting does not stop the invariant suite from continuing to run. Further, it uses default `runs` and `depth` values (256 and 500 respectively).

Recommendation: Consider adding to following configurations to the `foundry.toml` file:

```
[invariant]
runs = 512
depth = 1000
fail_on_revert = true
```

This could lead to a more robust invariant suite and identify some edge cases that need to be investigated.

Another improvement to the Invariant tests would be to add BTC based bounding to numbers used and possibly to add an invariant that the pBTC and stBTC balances should not exceed the total supply of BTC. This would force the invariant suite to use "real world" numbers and might surface issues (most likely improvements in the test suite).

Botanix: Acknowledged.

Spearbit: Acknowledged.

5.3.12 Incorrect bounding in invariants

Severity: Informational

Context: sBTCInvariants.t.sol#L113

Description: This is not entirely correct since `pBTC.balanceOf(actor)` could be less than `0.01 ether`. This causes `bound` to revert with `Max is less than min.`.

Recommendation: Adding `fail_on_revert` with forge invariant testing will surface this issue. A quick work around is to slightly alter and move sBTCInvariants.t.sol#L116 to before the bounding occurs.

Botanix: Acknowledged.

Spearbit: Acknowledged.

5.3.13 Tests won't run on Linux OS

Severity: Informational

Context: sBTCBase.sol#L6, sBTCInvariants.t.sol#L6

Description: There are a couple of case mismatches in the test file imports.

Recommendation: These should be corrected so tests run on Linux as well as Mac OS.

```
import {console2} from "forge-std/console2.sol";
```

Botanix: Fixed in commit cf2ef6ef.

Spearbit: Fix verified.

5.3.14 Potential Upgrade bug due to inconsistency between `stBTC.directDeposit` and `deposit` in 4626

Severity: Informational

Context: stBTC.sol#L205

Description: The new `directDeposit` function uses `msg.sender`, but the 4626 OZ implementation uses `_msgSender();`.

Recommendation: Use `_msgSender()` in `stBTC` so upgrades that include changes to the behavior with `_msgSender` will be applied appropriately for `directDeposit` as well as `deposit`.

Botanix: Acknowledged, but `msgSender()` should not change so will equal `msg.sender`.

Spearbit: Acknowledged.